LARGE EDDY SIMULATIONS OF TURBULENT FLOWS ON GRAPHICS PROCESSING UNITS: APPLICATION TO FILM-COOLING FLOWS

BY

AARON F. SHINN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:

Professor S. Pratap Vanka, Chair and Director of Research
Professor J. Craig Dutton
Assistant Professor Carlos Pantano
Professor Rizwan Uddin

Abstract

Computational Fluid Dynamics (CFD) simulations can be very computationally expensive, especially for Large Eddy Simulations (LES) and Direct Numerical Simulations (DNS) of turbulent flows. In LES the large, energy containing eddies are resolved by the computational mesh, but the smaller (sub-grid) scales are modeled. In DNS, all scales of turbulence are resolved, including the smallest dissipative (Kolmogorov) scales. Clusters of CPUs have been the standard approach for such simulations, but an emerging approach is the use of Graphics Processing Units (GPUs), which deliver impressive computing performance compared to CPUs. Recently there has been great interest in the scientific computing community to use GPUs for general-purpose computation (such as the numerical solution of PDEs) rather than graphics rendering.

To explore the use of GPUs for CFD simulations, an incompressible Navier-Stokes solver was developed for a GPU. This solver is capable of simulating unsteady laminar flows or performing a LES or DNS of turbulent flows. The Navier-Stokes equations are solved via a fractional-step method and are spatially discretized using the finite volume method on a Cartesian mesh. An immersed boundary method based on a ghost cell treatment was developed to handle flow past complex geometries. The implementation of these numerical methods had to suit the architecture of the GPU, which is designed for massive multi-threading. The details of this implementation will be described, along with strategies for performance optimization. Validation of the GPU-based solver was performed for fundamental bench-mark problems, and a performance assessment indicated that the solver was over an order-of-magnitude faster compared to a CPU.

The GPU-based Navier-Stokes solver was used to study film-cooling flows via Large Eddy Simulation. In modern gas turbine engines, the film-cooling method is used to protect turbine blades from hot combustion gases. Therefore, understanding the physics of this problem as well as techniques to improve it is important. Fundamentally, a film-cooling configuration is an inclined cooling jet in a hot cross-flow. A known problem in the film-cooling method is jet lift-off, where the jet of coolant moves away from the surface to be cooled due to mutual vortex induction by the counter-rotating vortex pair embedded in the jet, resulting in decreased cooling at the surface. To counteract this, a micro-ramp vortex generator was added downstream of the film-cooling jet, which generated near-wall counter-rotating vortices of opposite sense to the vortex pair in the jet. It was found that the micro-ramp vortices created a downwash effect toward the wall, which helped entrain coolant from the jet and transport it to the wall, resulting in better cooling. Results are reported using two film-cooling configurations, where the primary difference is the way the jet exit boundary conditions are prescribed. In the first configuration, the jet is prescribed using a precursor simulation and in the second the jet is modeled using a plenum/pipe configuration. The latter configuration was designed based on previous wind tunnel experiments at NASA Glenn Research Center, and the present results were meant to supplement those experiments.

To my wife Amanda and my children Claire and Addison.

Acknowledgments

I would like to thank my advisor and mentor, Dr. S. Pratap Vanka, for the opportunity to work with him and for the time he has invested in my education, research, and career development. I wish to thank NASA for awarding me a three-year Graduate Student Researchers Program (GSRP) Fellowship (grant number NNX08AY63H), where NASA Glenn Research Center (GRC) in Cleveland, Ohio was the sponsoring center. I would like to thank Mr. Russ Claus at NASA GRC for serving as my NASA Technical Advisor throughout my fellowship and for all the helpful discussions regarding the direction of my research. I thank Dr. Khairul Zaman for his insightful discussions about the experimental work at NASA GRC and for providing experimental data. I thank my colleague Mr. Rajneesh Chaudhary for many productive research discussions, for his implementation of the WALE SGS model, and his assistance with LES wall functions.

I thank the Computational Sciences and Engineering (CSE) program and the National Center for Supercomputing Applications (NCSA) at UIUC for partially supporting my research with a CSE Fellowship. In addition, I thank the MechSE department for providing supplemental financial support with multiple endowed fellowships.

I wish to convey my sincere gratitude to my doctoral committee (Dr. J. Craig Dutton, Dr. Carlos Pantano, and Dr. Rizwan Uddin) for their time and guidance in the development of this dissertation.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
CRVP	counter-rotating vortex pair
CUDA	Compute Unified Device Architecture
CV	control volume
DNS	Direct Numerical Simulation
DSSN	downstream spiral separation node
FLOPS	floating point operations
GPU	Graphics Processing Unit
IBM	Immersed Boundary Method
LBM	Lattice Boltzmann Method
LES	Large Eddy Simulation
MPI	Message Passing Interface
POSIX	Portable Operating System Interface for Unix
RANS	Reynolds-Averaged Navier-Stokes
SGS	sub-grid scale
SOR	successive over-relaxation
WALE	Wall-Adapting Local Eddy Viscosity model

List of Symbols

English

a	pressure coefficient
A	area
c_p	specific heat at constant pressure
C	convection term
C_s	Smagorinsky model constant
C_w	WALE model constant
d	diameter or distance
D	diffusion term
D_h	hydraulic diameter
F	flux
$\mathbf{F}$	body force term
G	filter function
H	height
$\mathbf{H}$	sum of convection and diffusion terms
I	turbulence intensity
k	thermal conductivity or turbulent kinetic energy
L	characteristic length
$\dot{m}$	mass flow rate
$\mathbf{n}$	normal vector
$ngrid$	number of grid levels for multigrid

p	pressure
Pr	Prandtl number
q_j	sub-grid scale heat flux vector
Q	volume flow rate
Re	Reynolds number
S	source term or span
S_{ij}	rate-of-strain tensor
t	time
T	temperature
$\mathbf{u}$	velocity vector
u_j	bulk jet velocity
u_τ	friction velocity
u_∞	freestream velocity
U	characteristic velocity
u, v, w	Cartesian velocity components
x, y, z	Cartesian coordinates

Greek

α	inclination angle of jet
δ	boundary layer height
δ_{ij}	Kronecker delta
Δ	filter width
Δt	time-step
$\Delta x, \Delta y, \Delta z$	mesh spacings
ϵ	turbulent dissipation
η	film-cooling effectiveness
$\overline{\eta}$	span-averaged film-cooling effectiveness

θ	rotation angle
μ	dynamic viscosity
ν	kinematic viscosity
ρ	density
τ	shear stress
τ_{ij}	sub-grid scale stress tensor
ϕ	scalar or flux limiter function
ψ	Gaussian random number
ω	vorticity or relaxation factor
Ω	domain
$\partial\Omega$	boundary of Ω

Subscripts

B	boundary point
$cell$	grid cell
$face$	face of a control volume
G	ghost point
i, j, k	cell indices or tensor indices
j	jet
jr	jet to ramp
LE	leading edge
M	mirror point
nb	neighboring value
p	plenum
r	ramp
rms	root mean square
t	turbulent

w	wall
$x+, y+, z+$	x-face, y-face, or z-face located at maximum value of x, y, or z, respectively, in a given control volume
$x-, y-, z-$	x-face, y-face, or z-face located at minimum value of x, y, or z, respectively, in a given control volume
∞	freestream condition

Superscripts

c	convective
d	diffusive
H	high-order
L	low-order
n	time level
$+$	wall units
$*$	non-dimensional quantity
$'$	fluctuating quantity
$\widehat{(\,)}$	provisional quantity or numerical approximation
$\overline{(\,)}$	time-averaged quantity
$\widetilde{(\,)}$	filtered quantity

Chapter 1

Introduction and Motivation

The research presented in this dissertation is in the area of Computational Fluid Dynamics (CFD). This study was divided into two parts, where the first part focused purely on computational science, and the second part focused on fundamental fluid dynamics. The computational science aspect involved implementing an incompressible Navier-Stokes solver on Graphics Processing Units (GPUs), with the goal of enhancing performance of existing CFD algorithms using this relatively new computing architecture. The fluid dynamics aspect involved using the GPU-based Navier-Stokes solver to study turbulent flow in film-cooling configurations, which has important applications for modern gas turbine engines. This chapter will provide an introduction and motivation of the research, and will conclude with an outline of the dissertation.

1.1 CFD using Graphics Processing Units

1.1.1 Overview

A novel contribution of this research is the exploration of a relatively new computing platform for CFD: the Graphics Processing Unit (or GPU). For years, the development of GPUs was driven by the need for fast graphics rendering, but recently there has been interest in the scientific computing community to use GPUs for general-purpose computation, such as in the numerical solution of PDEs. This interest has arisen out of the high-performance offered by GPU technology. Now manufacturers are releasing GPUs specifically designed for

scientific computing; an example is the NVIDIA Tesla C1060 shown in Figure 1.1, which delivers nearly 1 TeraFLOP in single precision and 78 GFLOPS in double precision. Even newer designs, such as the Tesla C2070 (using the Fermi architecture), deliver better double precision performance (over 500 GFLOPS). In addition to advances in hardware, there have been advances in the software used to program GPUs, such as the programming language called CUDA (Compute Unified Device Architecture), which was developed by NVIDIA. CUDA is like the C programming language, but with extensions added to access the GPU. This is a significant advance beyond previous GPU programming methods, where before the application had to be implemented using graphics rendering operations.

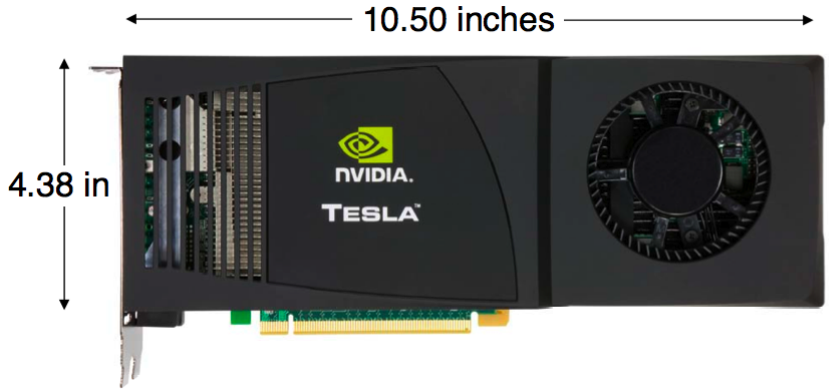


Figure 1.1: NVIDIA Tesla C1060 GPU.

For many years, numerical simulations have been successfully performed using single CPUs for serial execution of programs, CPU clusters for parallel execution across a network of processors (for example, using Message Passing Interface (MPI) [1]), or multiple/multi-core CPUs in a single machine (for example, using POSIX Threads [2] or OpenMP [3]). CPU performance has, however, increased only marginally in recent years, whereas GPU performance has increased dramatically. This trend is shown in Figure 1.2 using a performance comparison between Intel CPUs and NVIDIA GPUs over time. GPUs have delivered better single-precision performance since 2004, and better double-precision performance since 2008. Clearly, GPUs offer an attractive alternative compared with CPUs for scientific computing

applications.

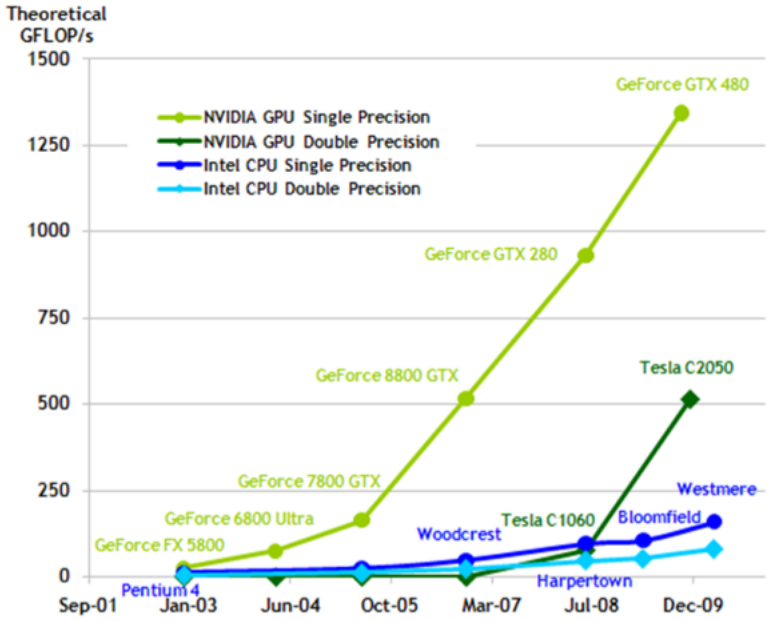


Figure 1.2: Performance of GPUs versus CPUs over time [4].

In the present research, the motivation for exploring GPU technology comes from the fact that CFD simulations can be very expensive computationally, especially for Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS) of turbulent flow. In LES the large, energy containing eddies are exactly resolved by the computational mesh, but the smaller scales (or sub-grid scales) are modeled. For DNS, all scales of turbulence are resolved, including the smallest dissipative (or Kolmogorov) scales. Both LES and DNS require large amounts of data storage and are very expensive computationally. For DNS, the requirements are more restrictive, since all scales must be captured by the computational mesh. For example, the following scalings relate the computational requirement of a DNS to the Reynolds number of the flow [5]:

$$\text{total number of grid nodes} \sim Re^{9/4} \tag{1.1}$$

$$\text{computational cost} \sim Re^{3} \tag{1.2}$$

where $Re = UL/\nu$, U is a characteristic velocity, L is a characteristic length, and ν is the kinematic viscosity of the fluid. It is clear from these scalings that simulating high Reynolds number flows using DNS is intractable even for massive computing clusters. Thus only low Reynolds number flows can be studied using DNS, which still require substantial computational resources.

1.1.2 Description of GPU Architecture

The architecture of a GPU is quite different than that of a CPU. A GPU is designed with more transistors dedicated to computation and less resources dedicated to data caching and flow control compared with a CPU, resulting in significant computational speed-up [4]. The GPU is designed to be a parallel processor by using massive multithreading, where a single thread can be thought of as the smallest unit of execution that executes instructions in a program. Instructions for the GPU are written in a "kernel" which is similar to a function in the C programming language. When a kernel is executed on a GPU, each thread executes the statements in that kernel, where each thread maps to a different element of data. Thus, the GPU architecture can be classified as SIMD (single-instruction, multiple-data) or SIMT (single-instruction, multiple-thread [4]). The number of threads needed for a particular kernel depends on the data size to be processed, since the threads map to the data element indices. Threads are organized into "blocks," and all blocks belong to a "grid" as shown in Figure 1.3. Before a kernel is executed on a GPU, the dimensions of the blocks and grid must be set explicitly by the programmer, as these are not automatically set by the GPU. It should be noted that the GPU is used as a co-processor in conjunction with the CPU. Typically, the "main" program executes on a CPU, and the GPU is utilized by launching kernels from the main program. Thus, usage of the CPU is not eliminated but rather is minimized.

A GPU contains multiprocessors, where each contains streaming processors or "cores." For example, the Tesla C1060 has 30 multiprocessors, each with 8 streaming processors,

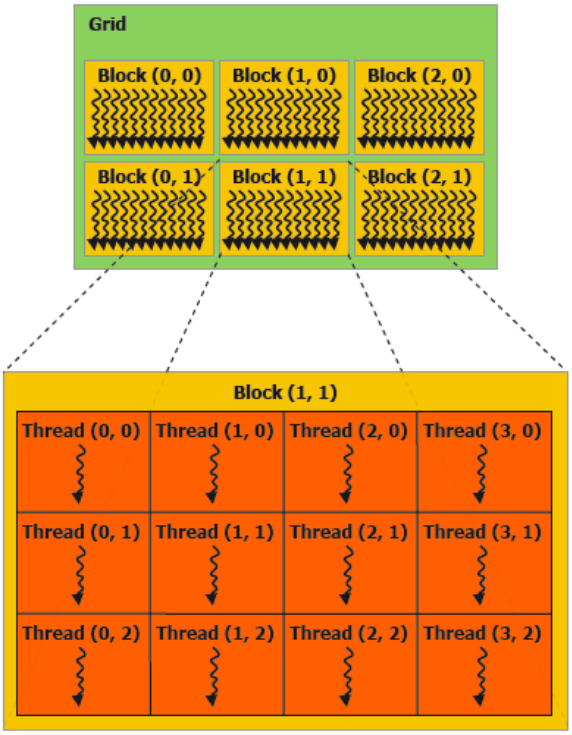


Figure 1.3: Thread hierarchy of the GPU [4].

and thus has a total of 240 streaming processors. The streaming processors are responsible for processing the thread blocks. When a block is processed, the threads in the block are divided into groups of 32 threads (called warps), and the streaming processor launches the threads in a warp in parallel [4]. The blocks are independent of each other and there is no synchronization among blocks, so the only way to ensure all blocks have executed is to wait until the kernel has finished and control has been returned to the main program.

In addition to the architectural differences between CPUs and GPUs, the memory bandwidth is another important difference. By memory bandwidth, we mean the rate (say in gigabytes per second) at which data is moved into and out of the random access memory (RAM). Modern GPUs have memory bandwidths an order of magnitude greater than CPUs; this is due to CPUs having to satisfy constraints of legacy applications and operating systems, which makes increasing memory bandwidth difficult, whereas GPUs have less legacy constraints resulting in more memory bandwidth [6]. This increase in memory bandwidth is

another factor contributing to the favorable performance of GPUs.

The memory spaces (RAM) of the CPU and GPU are separate, and explicit copy operations must be performed to move data to/from the GPU. The GPU has multiple memory spaces, as illustrated in Figure 1.4. Global memory is the largest memory space on the GPU (4 GB for the Tesla C1060). Each thread has its own registers and local memory, which are used for storing local variables declared within the kernel. All threads on a given block have access to the block's shared memory. However, a thread on a given block cannot access another block's shared memory; the only way for threads to access data corresponding to another block is to use global memory, which can be accessed by all threads on all blocks.

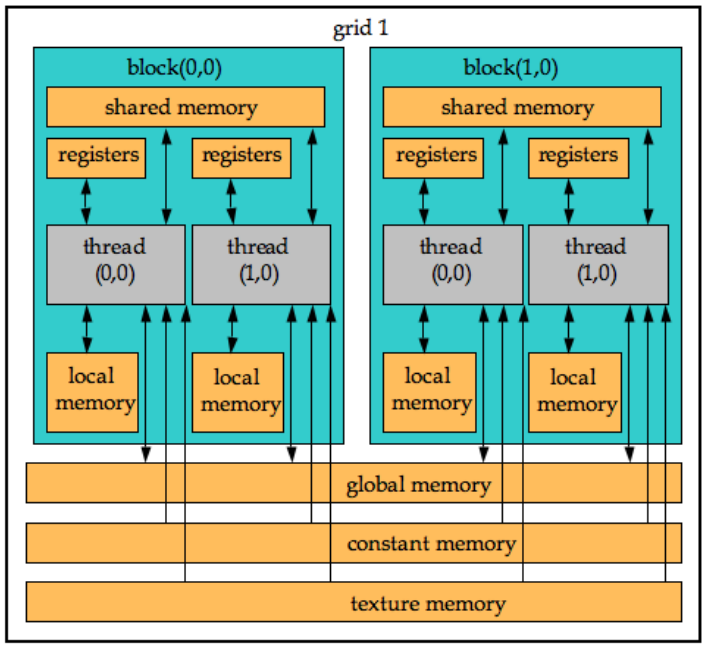


Figure 1.4: Memory model of the GPU.

Global memory is not cached, and thus has long access times. Shared memory is cached, and can be accessed much faster than global memory, but it is limited in scope (threads on a given block can only access their assigned shared memory) and size (only 16 kB per block on modern GPUs). In order to use shared memory, data must first be transferred from global memory to shared memory; computations are then performed using shared memory

and the results are written back to global memory. This introduces extra computational complexity in the algorithm, but this can be offset by the potential gains of using a cached memory space. This benefit is realized if the data loaded into shared memory are reused many times during kernel execution. While global memory has the disadvantage of having longer access times than shared memory, the algorithm becomes considerably simpler using global memory. In addition, if the data are not reused many times during kernel execution, then global memory should be used as shared memory would not provide much benefit. The constant memory is a read-only cached memory space that can be accessed by all threads, which can be used to store a look-up table of constants; however, this space is small (only 64 kB). The texture memory is another read-only cached memory space that can be accessed by all threads, which offers avenues for performance optimization. For example, data structures in global memory can be read through texture memory via texture fetching, which decreases the access time.

1.1.3 Description of GPU Programming

Writing a program for a GPU is quite different than for a CPU. For a CPU program, instructions in the program are executed sequentially by a single thread. For a GPU program the instructions are executed by multiple threads, where each thread is unique and assigned to a unique element of data to be computed. To elucidate the difference between CPU and GPU programming, consider the code samples in Table 1.1, where the CPU code is written in C and the GPU code is written in CUDA. Both programs perform the same task, which is to call a routine that performs an element-wise sum of the ten elements of `a` and `b` and writes the result to `c`. In the CPU code, this is performed by calling a function that performs the sum using a for-loop, which is executed by a single thread of the CPU. In the GPU code, an "execution configuration" is made, where the dimensions of the blocks and grid are set. The dimensions of the block dictate how many threads are needed per block and the dimensions of the grid dictate how many blocks are needed.

Table 1.1: Code samples for the CPU and GPU to illustrate different programming philosophy.

CPU code written in C	GPU code written in CUDA

`int main(void)`
`{`
`  ...`
`  function(a,b,c);`
`  ...`
`}`
 	
`void`
`function(int *a,int *b,int *c)`
`{`
`  for(i=1; i<=10; i++)`
`  {`
`  c[i] = a[i] + b[i];`
`  }`
`}`
	`int main(void)`
	`{`
	`  ...`
	`  dim3 block(10,1,1);`
	`  dim3 grid(1, 1);`
	`  kernel<<<grid, block>>>(a,b,c);`
	`  ...`
	`}`
	
	`__global__ void`
	`kernel(int *a,int *b,int *c)`
	`{`
	`  i = threadIdx.x + 1;`
	`  c[i] = a[i] + b[i];`
	`}`

For this trivial example, the block size is set with 10 threads in the x-direction and unity in the y- and z- directions. In addition, the grid is set using only a single block. Thus the total number of threads in the GPU grid is equal to 10, which is the total number of elements in our data structures `a`, `b`, and `c`. This is done so that the threads will map one-to-one with the data elements. The kernel is launched using the `<<< >>>` syntax and the instructions in the kernel are executed by all the threads. The variable `threadIdx.x` is the thread index, which starts at zero by default; thus the index range will be from zero to nine. This index is offset by unity so that it will access the correct values in the data structures. In this way, each thread is mapped to a unique index of the data structures, and the operation is performed in parallel. The fundamental difference between the two codes is that the loop has been eliminated in the GPU code, allowing the code to be executed in parallel by a batch of threads on the GPU rather than a single thread on the CPU. This example only used a single block since the data size was small, but larger problems will use more threads per block and have a large amount of blocks per grid to accommodate the large data size.

The above example problem was easy to make parallel, since the sum was element-wise and did not depend on other data. However, for more complex algorithms (such as a Navier-Stokes solver) the implementation is not so simple, and the algorithms will need to be changed to suit the parallel nature of the GPU. Note that this level of parallelism is different than previous forms of parallel computing. For example, if one were to use MPI to solve a PDE in parallel using a CPU cluster, a standard approach would be to perform a domain decomposition on the PDE domain, and assign each sub-domain to a different CPU. The CPUs would communicate boundary data between the sub-domains every time the solution was updated. Instead of updating the solution in the entire domain, the PDE solver for a given CPU would update only the assigned sub-domain, where the algorithms still execute in serial fashion for that sub-domain. Thus data parallelism exists only on the sub-domain level. This is not the case with GPU programming, where data parallelism must exist down to the individual mesh points. If this level of parallelism does not already exist in a given

algorithm, then the numerical methods will need to be changed to suit the GPU.

Once an algorithm has been implemented on a GPU, it is not guaranteed to yield one or two orders of magnitude of speed up automatically. Careful performance optimization is required to achieve maximum throughput. For example, memory copies from the CPU to GPU (and vice-versa) are extremely expensive, and should be minimized. The minimum would be copying initial data to the GPU and copying the final data back from the GPU, but this author recommends occasionally copying information about residuals (say mass and pressure residuals) to the CPU so that they may be printed to the screen. As mentioned previously, global memory access is expensive on a GPU. If a value from global memory is accessed multiple times during kernel execution, an easy way to decrease memory latency is to store the value to a register (local variable) first, and then use the register value from then on.

Another way to decrease global memory access time is to arrange the data structures such that accesses are *coalesced*. For a comprehensive explanation of the requirements for memory coalescing, the reader is referred to [7]. Generally, coalescing is achieved if sequential threads of a half-warp (16 threads) access sequential elements in memory, with the result that memory loads of the multiple data elements are batched into a single load, saving considerable time. This may not be possible if using indirect addressing, which is common in some CFD codes. To circumvent this, texture memory can be used as a substitute, where data in global memory is accessed through a texture fetch. Another important optimization is the grid size and block size that are set in the execution configuration. The grid size (number of blocks in a grid) should be greater than the number of multiprocessors to ensure that all multiprocessors have at least one block to execute. The block size (number of threads per block) should be set such that it is a multiple of the warp size. This has two advantages: it will help promote memory coalescing and prevent wasting compute time on warps that are not fully populated [7]. This author has found that GPU performance can be very sensitive to block size, and it is recommended that one should iterate on the block size using test runs

to determine the optimal size before running long simulations.

1.2 Film-Cooling

Now that the computational side of the study has been discussed, we turn our attention to the fluid mechanics aspect, which is the study of turbulent flow in film-cooling configurations.

1.2.1 Overview

In gas turbine engines, the temperature of the combustion gases leaving the combustor and entering the first stage of the turbine typically exceeds the melting point of the turbine blade material. Thus, protecting the blades from hot combustion gases is crucial to prevent material failure. A standard method used to cool the blades is called *film-cooling*. In the film-cooling method, air is bled from the compressor of the engine and transported through internal passages until it reaches the interior of the turbine blades (a plenum cavity). The air then travels from the interior of the blade through inclined tubes leading to the blade surface. The air exits the tubes and is injected into the hot cross-flow but near the blade surface. The jets of air are meant to create a film of coolant that covers the blade surface to protect it from the combustion gases. The coolant is injected at an inclination so that it remains close to the blade surface to maximize coverage and minimize penetration into the cross-flow. The practical importance of this problem is motivation for understanding the flow physics of film-cooling and for investigating methods to optimize it.

Figure 1.5 shows a turbine blade from a gas turbine engine that uses film-cooling, where the cooling holes are equally spaced and arranged in rows. While this is a practical geometry to study, most research has focused on simpler configurations so that more general physics can be extracted. An example of a simplified configuration is shown in Figure 1.6, which is a typical film-cooling configuration used for laboratory and numerical experiments. In this configuration, coolant air flows into a plenum, which is a large stagnation-like chamber,

which feeds the coolant into an inclined tube. The exit of the tube is at the flat plate surface, where the coolant exits the hole and forms a jet that interacts with the freestream cross-flow.

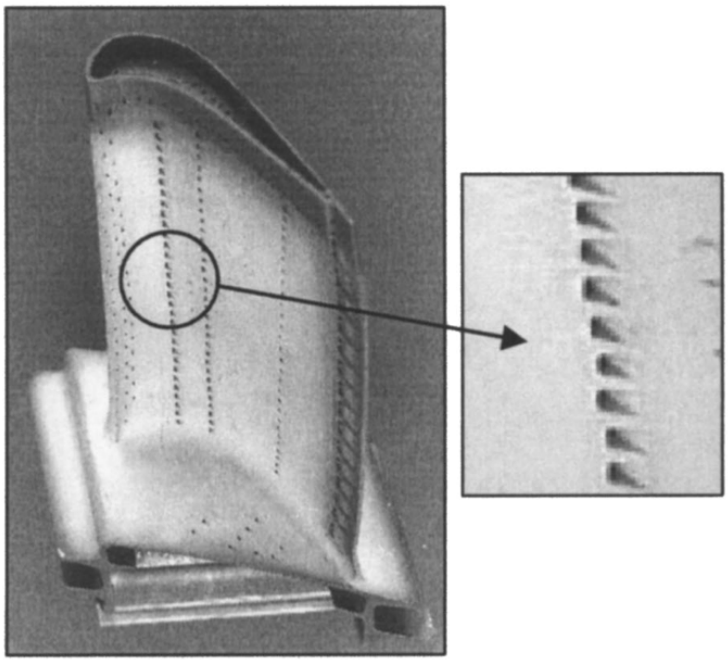


Figure 1.5: Film-cooled turbine blade [8].

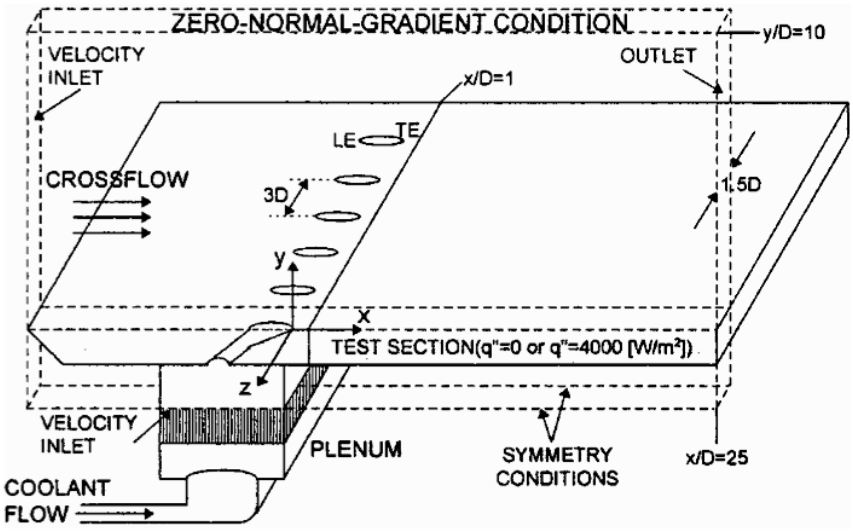


Figure 1.6: Typical film-cooling configuration for laboratory and numerical experiments [9].

1.2.2 Physics of Film-Cooling: a Jet in a Cross-flow

Fundamentally, a film-cooling configuration is a jet in a cross-flow (JICF), where the jet exits at an inclination to the cross-flow. Several previous works have explored the nature of a JICF where the jet exits normal to the cross-flow; notable works include the experiments of Andreopoulos and Rodi [10], Fric and Roshko [11], Peterson and Plesniak [12], Su and Mungal [13], Zaman and Foss [14] and numerical efforts of Muppidi and Mahesh [15, 16] and Yuan et al. [17]. The salient features of a JICF include shear-layer vortices at the leading edge of the jet, a counter-rotating vortex pair (CRVP) in the jet (also known as kidney vortices owing to their shape), horseshoe vortices around the base of the jet, and wake vortices downstream of the jet. These coherent turbulent structures are illustrated in Figure 1.7. These flow features are also found in film-cooling flows, and will be identified in the present work.

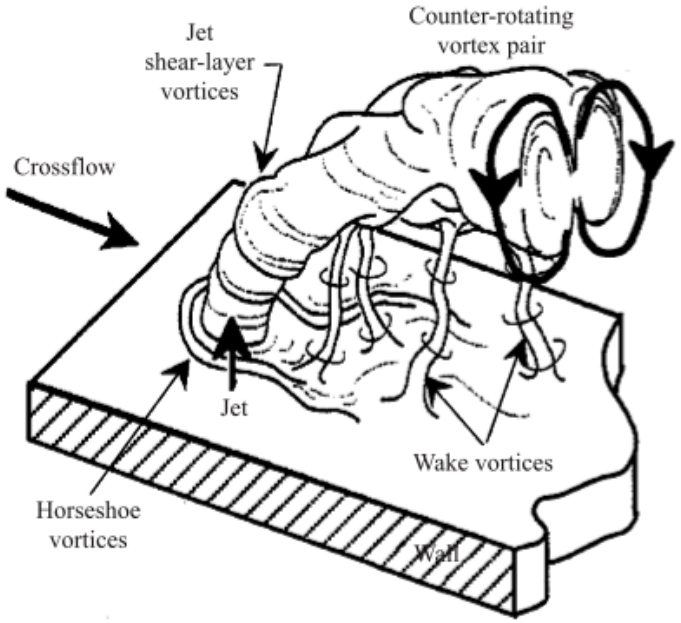


Figure 1.7: Illustration of a jet in a cross-flow with coherent structures identified [12].

The shear-layer vortices are generated by a Kelvin-Helmholtz instability at the interface between the jet and cross-flow. The horseshoe vortices are caused by the cross-flow boundary layer impinging on the jet flow, which results in the boundary layer flow "rolling up" at the

base of the jet. The wake vortices are generated downstream of the jet exit and are due to the entrainment of the cross-flow boundary layer into the jet, and appear similar to tornado-like structures. The CRVP can be observed in the jet cross-section downstream of the jet inflow. This feature is not recognizable instantaneously, but is readily seen in the time-averaged flowfield. As the jet flow is bent by the cross-flow, the CRVP forms within the jet in the near-field and persists in the far-field. The exact origin of the CRVP is still debated, and an excellent review of this issue is given by Peterson and Plesniak [12].

1.2.3 Techniques for Improving Film-Cooling

A problem with the film-cooling method is that the jet of coolant tends to lift off the blade surface, resulting in decreased film-cooling effectiveness. The lifting of the coolant is caused by vortex induction from the CRVP. Previous studies have focused on ways to reduce the detrimental effect of vortex induction and enhance cooling effectiveness. The main advancement for improving film-cooling has been shaping the coolant holes, where instead of a round hole leading to the surface the cross-sectional area is increased near the surface and is shaped similar to a diffuser. This results in a reduction in jet velocity, causing reduced jet penetration into the cross-flow and allows the jet to remain attached to the surface. In addition, the shaping results in increased lateral surface coverage of the coolant. By allowing the coolant to remain attached to the surface and increasing the lateral spread of the jet, the overall cooling effectiveness is increased. This method has become prevalent in modern gas turbine engines. Shaped film-cooling holes have some disadvantages, however, including increased manufacturing costs and thermal barrier coating (TBC) limitations [18]. A review of the research in shaped film-cooling is given by Bunker [8].

In addition to shaped film-cooling, other methods have been studied to enhance cooling effectiveness. These include pulsing the jet [19, 20, 21], including micro-ramps near the jet hole [22, 23] and using an "antivortex" film-cooling hole design [18, 24]. These research studies will be reviewed in Chapter 2. In the present study, micro-ramp vortex generators

will be investigated as a technique to improve film-cooling.

1.2.4 Numerical Simulation of Film-Cooling: The Need for DNS and LES

Many numerical investigations of film-cooling flows have used the Reynolds-Averaged Navier Stokes (RANS) equations (see, for example [25, 26, 27, 28, 29]). In this approach, the Navier-Stokes equations are time-averaged, so that the resulting variables solved for in the numerical solution are the mean quantities, rather than the instantaneous quantities. In addition, the averaging of the momentum equation produces the Reynolds stress tensor, which must be modeled (using, for example, algebraic or two-equation turbulence models). While computationally efficient, the RANS approach has been shown to provide poor predictions of film-cooling flows. For example, previous RANS-based studies have shown that this method tends to overpredict the centerline film-cooling effectiveness [27], overpredict the vertical penetration of the jet into the cross-flow [28], and underpredict the lateral spread of the jet. The RANS approach is useful for qualitative descriptions of film-cooling flows, but is unreliable for extracting detailed flow physics.

The film-cooling problem is a complex, three-dimensional, unsteady flow. In order to accurately resolve the flow features spatially and temporally, it is more appropriate to use Large Eddy Simulation (LES) or Direct Numerical Simulation (DNS). Previous work has indicated that much better predictions of the physics are possible using LES and DNS. Examples of previous film-cooling studies using LES include [30, 31, 32] and using DNS include [19, 33]. These works will be reviewed in detail in Chapter 2. In the present work, film-cooling flows will be investigated using LES.

1.3 Contributions of Research

The following list summarizes the contributions of the present research.

- To explore GPUs for CFD applications, a 3D incompressible Navier-Stokes solver was implemented on a single GPU. The code (called CU-FLOW[1]) was written using CUDA (Compute Unified Device Architecture), which is a programming paradigm developed by NVIDIA. The GPU-based solver was validated for laminar and turbulent flows in basic geometries. The GPU-based solver has shown impressive speedup (over an order-of-magnitude) compared to a similar CPU-based flow solver. For the present research, the single GPU solver was the "production code" used for the LES of film-cooling flows.

- The GPU-based Navier-Stokes solver was extended to multiple parallel GPUs to investigate the potential for increasing speed-up. This was tested using a workstation with a quad-core CPU and four parallel GPUs. The multi-GPU code was implemented by using POSIX threads for the multi-core CPU, where each POSIX thread is assigned a GPU to control. The code was validated for a 3D driven cavity and has shown good speed-up scaling relative to the single GPU code.

- Large Eddy Simulations of film-cooling flows were performed to study the fundamental fluid mechanics and heat transfer characteristics. In addition, a micro-ramp vortex generator was placed near the film cooling jet to investigate if this device counter-acts the detrimental effects of jet lift-off and provides increases in cooling effectiveness. This is the first reported LES of film-cooling flows using micro-ramps.

- A 3D ghost cell method was developed to handle flow past complex geometries. This method was implemented to improve the solution near obstacle boundaries, since the Cartesian mesh is not boundary-fitted. The method was designed to be general, so that in principle any complex geometry can be immersed in the Cartesian mesh by supplying vertices of a surface triangulation representing the obstacle surface. In the present study, this method is used to represent the micro-ramp surface.

[1]the name is short for "Cartesian Unsteady Flow" and also for "Champaign-Urbana Flow"

1.4 Summary and Outline of the Dissertation

To summarize, the present research involves developing a Navier-Stokes solver for GPUs, which is motivated by the desire to explore new ways to improve computational performance of CFD algorithms. This solver is then used to study the physics of film-cooling flows, and in particular investigate the effect of including a micro-ramp vortex generator near the film-cooling jet. This work is motivated by the practical need to understand and improve film-cooling flows in gas turbine engines.

Next, a literature review will be presented in Chapter 2 describing developments in both GPU-based CFD and film-cooling flows. The details of the numerical methods used in the present CFD simulations are presented in Chapter 3, along with the GPU implementation of these methods. Results of validation cases are presented in Chapter 4, which include laminar and turbulent flows in basic geometries. In Chapters 5 and 6, two film-cooling studies are presented. In both cases, a configuration similar to Figure 1.6 is used. The primary difference between the studies is how the jet inflow conditions are handled. In Chapter 5, the plenum and pipe are not modeled, and a precursor simulation is used to provide the jet inflow boundary condition and results are compared to previous DNS results. In Chapter 6, the pipe and plenum are included in the simulation, and results are compared with experiments. Lastly, the thesis concludes with Chapter 7 that presents the final conclusions and recommendations.

Chapter 2

Literature Review

This chapter presents a review of the literature in the three primary areas of focus for the present study: fluid dynamics simulation using GPUs, jets in cross-flow, and immersed boundary methods.

2.1 CFD using Graphics Processing Units

Several previous works have shown the promise of using GPUs for CFD applications. Before the advent of CUDA, programmers had to cast their applications in terms of graphics processing operations. Early work of this type was done by Scheidegger et al. [34], where they presented a GPU implementation of the SMAC method (Simplified Marker And Cell) to solve the 2D, incompressible Navier-Stokes equations. They used structured grids (where a staggered arrangement of the variables was used) and their approach works with obstacles in the domain and with various boundary conditions. Central differences and a hybrid donor cell scheme were used in their approach. Texture memory was used to store the data structures; for example, floating-point textures called pixel buffers (or "pbuffers") were used to store the velocity fields. The Jacobi iteration scheme was used as a fragment program for the solution of the pressure-Possion equation. They note that over 95 percent of the program execution time was used in solving the pressure-Poisson equation. In their study, two GPUs were tested: a GeForce FX 5900 (NV35) and a GeForce 6800 Ultra (NV40). The CPU used was a 2 GHz Pentium IV. The CPU performed better than the GPU only when a very small mesh was used, and this occurred when using the NV35. They explain that

convergence is rapid in this case and that the pbuffer switches "probably overshadowed" the GPU parallelism. Their approach was, on average, approximately 16 times faster than the CPU version. They studied a variety of flows, such as a lid-driven cavity, rising smoke at high Reynolds number, and flow past the outline of a car at low Reynolds number.

Elsen et al. [35] used a GPU to simulate the inviscid flow in simple and complex geometries by numerically solving the compressible Euler equations. Compared to the CPU, they achieved GPU speed-ups of over 40 times for simple geometries and 20 times for complex geometries. The complex geometries consisted of a NACA 0012 airfoil and a hypersonic vehicle at Mach 5. The maximum speed-up for the NACA 0012 airfoil was 17.6 and the maximum speed-up for the hypersonic vehicle was 20.2. Their comparisons were based on a 2.4 GHz Intel Core 2 Duo (single core used) and NVIDIA 8800GTX. They used the Navier-Stokes Stanford University Solver (NSSUS), which is capable of solving the 3D Unsteady Reynolds Averaged Navier-Stokes (URANS) equations. This code uses the finite-difference method with a vertex-centered solution on multi-block meshes. Temporal evolution of the solution toward a steady-state was accomplished using an explicit five-stage Runge-Kutta scheme, which used modified coefficients to give a maximum stability region. The code also incorporated a geometric multigrid scheme to accelerate convergence. For their study, only the steady solution of the compressible Euler equations was sought. They used the BrookGPU language to implement NSSUS (which was originally in Fortran) on a GPU and report that it required around 4 months to implement the code for the GPU. They note that the GPU results were accurate (within 5 to 6 significant digits) compared to the original single-precision CPU code.

Brandvik and Pullan [36] presented results for 2D and 3D Euler solvers implemented on the GPU. Their original implementation on the CPU of the Euler solvers was written in Fortran, and the GPU version of the Euler solvers yielded the same results as the CPU counterpart. They used the Euler solvers to simulate turbine flows: the 2D solver was used to simulate the flow through a transonic turbine cascade and the 3D code was used to

simulate secondary flow development in a low speed linear turbine cascade. The 2D solver was programmed for the GPU using the BrookGPU language, and performed 29 times faster than the CPU version. They also used BrookGPU for the 3D solver, which performed only 3 times faster than the CPU version. A CUDA implementation of the 3D solver yielded better performance with a speed-up of 16 over the CPU. A 2.33 GHz Intel Core 2 Duo processor was used for the CPU solvers, where only a single core was utilized. The 3D CUDA solver used an NVIDIA 8800GTX graphics card and the 2D and 3D BrookGPU solvers used an ATI 1950XT graphics card. Their 2D and 3D codes solved the compressible Euler equations using the finite-volume method with structured grids, where the variables were stored at the cell vertices. The spatial derivatives were discretized via second-order central differences and the temporal derivatives were discretized to first-order accuracy. There was no multigrid method employed in their approach. Their algorithm was designed such that the preprocessing and postprocessing was performed on the CPU and the calculation of the solution was performed on the GPU.

Cohen and Molemaker [37] present a GPU implementation using CUDA for solving the incompressible Navier-Stokes equations with the Boussinesq approximation. They present results for the simulation of the Rayleigh-Benard convection problem and compare their GPU implementation to a multithreaded Fortran solver running on an eight-core CPU. Using double precision, the GPU-based solver was approximately eight times faster. Shinn and Vanka [38] were the first to implement the SIMPLE algorithm on a GPU. Using CUDA, they wrote a 2D solver with multigrid Full Approximation Scheme (FAS) used to accelerate convergence of all flow variables (u, v, and p). The code was tested for the benchmark 2D driven cavity problem and compared to a CPU version of the code written in Fortran. It was found that the speedup scales with the problem size. For a problem size of 512 x 512 grid cells, the GPU was an order-of-magnitude faster than the CPU for a range of Reynolds numbers. Steady-state calculations of driven cavity flow with 4096 x 4096 could be performed in a minute of GPU time.

Shinn et al. [39] performed one of the first Direct Numerical Simulations using GPU hardware. The fractional-step method with finite volume spatial discretization was used to solve the incompressible Navier-Stokes equations, and was implemented on a GPU using CUDA. A geometric multigrid method was used to accelerate the pressure-Poisson solution. They simulated turbulent flow in a square duct at a bulk Reynolds number of 5480 using a mesh resolution of 26.2 million cells. This problem was selected not only to validate the GPU-based solver but also to test the capability of the GPU, as this was the largest problem that could fit on a single Tesla C1060. The salient features of this canonical flow were captured and compared well with previous data. The GPU-based solver was over an order-of-magnitude faster compared with the CPU-based version. Chaudhary et al. [40] extended this solver to include magnetohydrodynamics and used it to study the magnetic field effects on turbulent flow in a square duct. Direct Numerical Simulations were performed at a bulk Reynolds number of 5500 at different Hartmann numbers to vary the magnetic field.

Thibault and Senocak [41] presented the first implementation of a 3D incompressible Navier-Stokes solver on multiple GPUs. Using CUDA, a fractional-step procedure was used to solve the equations and the pressure-Poisson equation was solved using Jacobi iteration with no multigrid scheme. The spatial terms were discretized with second-order accurate central differences and an explicit, first-order accurate Euler scheme was used for temporal advancement. They validated their GPU implementation and assessed speedup via the problem of laminar flow in a lid-driven cavity. This was the only problem they tested with their GPU solver. Their GPU solver was 13 times faster using a single NVIDIA Tesla C870 GPU and 21 times faster using two NVIDIA Tesla C870 GPUs compared with their CPU solver running on a single core of a 3.0 GHz Intel Core 2 Duo processor. Their GPU solver was found to be 100 times faster using an NVIDIA Tesla S870 server (4 GPUs) when compared with their CPU solver running on a single core of a 2.4 GHz AMD Opteron processor.

Griebel and Zaspel [42] were the first to implement a two-phase Navier-Stokes solver on a GPU, where they used a level set technique for the two-phases and a fractional step

method to solve the Navier Stokes equations. They implemented the solver on multiple GPUs and communicated GPU data between CPUs using Message Passing Interface (MPI). They ported a solver for the pressure-Poisson equation (a Jacobi-preconditioned conjugate gradient solver) and the level set reinitialization to the GPU. In order to minimize the overhead from data communication, they exploited the asynchronous communication feature of GPUs, where data can be copied while computations are being performed. This can effectively hide the communication time. This was done using "streams" where one stream managed communication while the other managed computation. On a single GPU, their Poisson solver and level set reinitialization were accelerated by a factor of 16.2 and 8.6, respectively, compared to a single CPU. Using eight GPUs, these factors increased to 115.8 and 53.7, with close to linear scaling.

Other than finite-difference and finite-volume methods, there has been progress in implementing the Lattice-Boltzmann Method (LBM) for simulating fluid flows on GPUs. Early work by Li et al. [43] provided an implementation of the LBM on a GPU. Their implementation was capable of dealing with complex boundaries (both moving and deformable), which were managed using a voxelization algorithm. All calculations were performed on a GPU in real time and their simulations were second-order accurate in time and space. Visualization of the flow was achieved by placing colored particles in the flow and allowing them to move with the velocity field. Their LBM code was programmed in Cg and OpenGL and used an NVIDIA GeForce FX 5900 Ultra GPU. The CPU used was a 2.53 GHz Pentium IV. They simulated a number of complex geometries on the GPU, such as a vase, a sphere, and a swimming jellyfish. It was found that their GPU implementation was 8 to 15 times faster than the CPU counterpart.

Tölke [44] used the 2D Lattice Boltzmann Method on a GPU by programming in CUDA. The implementation was tested by considering the fluid flow through a porous medium, which consisted of a grid of 324 circular cylinders, equally-spaced in the horizontal and vertical directions. The GPU implementation was over 10 times faster relative to the CPU. Peng

et al. [45] developed a 3D Lattice Boltzmann method algorithm for a GPU using CUDA. They compared a NVIDIA GPU (GeForce 8800 GTS) with an AMD CPU (Sempron 3500+) and found that the GPU performed 8.76 times faster than the CPU. They also developed a version of the LBM code for the Cell processor in the PlayStation3, which was programmed using the IBM Cell Software Development Kit, version 2.1. They tested their LBM code on a cluster of nine PlayStation3 systems, where communication among systems was achieved using Message Passing Interface (MPI). The performance of the PlayStation3 cluster was compared to a Xeon cluster, which had 256 compute nodes. Each node had two 2.8 GHz processors. The PlayStation3 cluster was found to be 11.02 times faster than the Xeon cluster. As an example of a complex geometry, they used their LBM implementation for the simulation of fluid flow through fractured glass.

Recently, Marsh [46] used CUDA to implement a hybrid molecular dynamics/Lattice Boltzmann method on GPUs. Flow through a nano-scale straight channel and a nano-scale bellow channel were investigated. In the hybrid method, a molecular dynamics solver was used in the near-wall region and a Lattice Boltzmann solver was used away from the wall. The GPU provided a speed-up factor of 5-10 for the molecular dynamics solver and 50-75 for the Lattice Boltzmann solver compared to a CPU. The large speed-up for the Lattice Boltzmann method is indicative of the fact that this method is easier to parallelize (or, strictly speaking, multithread) compared to molecular dynamics. As an extension of this work, Sahu and Vanka [47] implemented a two-phase LBM on a GPU and observed a speed-up factor of 25 over a CPU.

2.2 Jets in Cross-Flow and Film-Cooling Flows

Extensive previous research exists on jets in cross-flow, for both normal jets in cross-flow (where the jet flow is perpendicular to the cross-flow) and inclined jets in cross-flow (film-cooling flows). The following review discusses a sample of this body of literature, including

some seminal works and research relevant to the present study. For supplementary reading, the reader is referred to the review of jets in cross-flow by Margason [48], the film-cooling review by Bogard and Thole [49], the review of early film-cooling research by Goldstein [50], and the film-cooling CFD bibliography by Kercher [51].

2.2.1 Normal Jets in Cross-flow

A jet flowing normal to a cross-flow is an important fundamental problem in fluid mechanics, and has received much attention over the years. Andreopoulos and Rodi [10] performed an experiment on a circular, turbulent jet issuing normally into a cross-flow, where the cross-flow moved along a flat plate and the jet exited at the flat plate surface. They considered velocity ratios of 0.5, 1, and 2 at Reynolds numbers based on the jet velocity and pipe diameter of 20500, 41000, and 82000, respectively. A three-sensor hot-wire probe was used to measure mean and fluctuating velocity components. A counter-rotating vortex pair (CRVP) was observed in the jet, and they explain that the CRVP results from a reorientation and stretching of the vorticity from the pipe flow of the jet (dominant mechanism at lower velocity ratios) and also from shearing at the interface between the jet and cross-flow (dominant mechanism at higher velocity ratios). In addition, it was found that the CRVP was stronger at higher velocity ratios and persisted longer in the jet far-field.

Fric and Roshko [11] experimentally investigated a turbulent jet issuing normally into a uniform cross-flow, where the focus of the study was on the formation of wake vortices. The jet to cross-flow velocity ratios ranged from 2 to 10 at cross-flow Reynolds numbers from 3800 to 11400. They utilized smoke-wire flow visualization for the wake and single hot-wire probes for velocity measurements. The important finding in their study was the explanation for the formation of wake vortices. They note that there is no vortex shedding from the jet into the wake (such as vortex shedding from a circular cylinder). Rather, the cross-flow around the jet created an adverse pressure gradient at the wall, near the trailing edge of the jet. This caused the boundary layer to separate from the surface, and "tornado-like" vortices

were formed alternately on either side of the jet. The base of the vortices was located in the boundary-layer and the top terminated in the jet.

Su and Mungal [13] studied a circular jet in a cross-flow experimentally using planar laser-induced fluorescence and particle image velocimetry (PIV), where the emphasis of the study was on scalar mixing. They examined two configurations: 1) jet exit coplanar with the flat plate surface and 2) jet pipe protruded into the cross-flow. The jet to cross-flow velocity ratio was 5.7 and Reynolds number based on the jet velocity and jet diameter was approximately 5000. Air was used as the cross-flow fluid and nitrogen (seeded with acetone vapor) as the jet fluid. The ratio of the jet fluid density to the cross-flow fluid density was 1.10. It was found that the mean scalar field using the protruding jet pipe was similar to the results obtained using the coplanar jet exit. The velocity profile at the jet exit was found to be more influential in the evolution of the mixing field than the velocity ratio or jet pipe position. Using the scalar variance and magnitude of turbulent scalar fluxes, it was found that the intensity of the mixing was initially larger at the jet leading-edge, but later became larger at the trailing-edge.

Yuan et al. [17] performed a numerical study of a circular jet in a cross-flow using Large Eddy Simulation with a dynamic SGS model. They considered Reynolds numbers of 1050 and 2100 based on the cross-flow velocity and jet diameter with two jet to cross-flow velocity ratios of 2.0 and 3.3. The cross-flow velocity boundary condition was prescribed as a laminar boundary layer profile. The computational domain contained a one-diameter section of the jet pipe below the flat plate to allow the jet profile to adjust to the cross-flow. The inflow boundary condition for the jet was prescribed using a pipe flow simulation that was run at the same time as the jet in a cross-flow simulation, where the instantaneous velocity field from the pipe was extracted at each time-step for the jet. They compared the turbulent statistics from their LES versus experimental measurements of Sherif and Pletcher [52] and reasonable agreement was obtained. They proposed a new formation mechanism for the CRVP based on their simulation. They noted that quasi-steady vortices (called "hanging"

vortices by the authors) form on the lateral edges of the jet. When these vortices encounter an adverse pressure gradient at the trailing-edge of the jet they break down into a pair of vortices aligned with the jet trajectory, which is the CRVP.

Muppidi and Mahesh [16] performed a Direct Numerical Simulation of a circular turbulent jet in a laminar cross-flow. The cross-flow moved along a flat plate and the jet exit was coplanar with the plate. The Reynolds number based on the jet velocity and jet diameter was 5000 and the jet to cross-flow velocity ratio was 5.7. The computational domain included two diameters of pipe below the flat plate to allow the jet profile to adjust to the cross-flow. The cross-flow velocity boundary condition was prescribed as a laminar Blasius boundary layer. The jet inflow condition was prescribed using data from a precursor simulation of turbulent pipe flow. Instantaneous velocity data were stored from the precursor for a period of time and then were interpolated onto the jet inflow plane. They compared their DNS results to the experiment of Su and Mungal [13] and good agreement was observed. It was found that peak production of turbulent kinetic energy was located near the jet leading-edge and peak dissipation was located near the jet trailing-edge. The turbulent kinetic energy budget revealed that the flow is not in turbulent equilibrium. They concluded that the failure of the RANS approach at simulating jets in cross-flow is related to the steep gradients in the flowfield variables, complex budget profiles at the jet exit, and non-equilibrium behavior of the flow.

2.2.2 Inclined Jets in Cross-Flow (Film-Cooling Flows)

A number of fundamental experimental and numerical studies have been performed to understand the physics of film-cooling flows. Sinha et al. [53] performed an experimental film-cooling study using a flat plate with a row of 7 jet holes spaced 3 diameters apart in the spanwise direction, where each jet hole was inclined at 35 degrees to the flat plate. Cryogenically cooled air was injected through the jet holes into the cross-flow, and the density ratio of the coolant to cross-flow was varied from 1.2 to 2.0. A surface thermocouple ar-

rangement was used to measure temperatures at the wall. They observed that the jet can remain attached to the wall surface, detach and then reattach, or become fully detached. As the momentum flux ratio was increased, the jet detached from the wall surface. In addition, they found that decreasing the density ratio and increasing the momentum flux ratio caused reduced spreading of the jet, resulting in reduced spanwise-averaged effectiveness. It should be noted that the data set from this work has become popular for validating CFD codes for film-cooling problems.

In the first part of a four-paper computational study, Walters and Leylek [25] studied the physics of film-cooling in a configuration with streamwise injection of the jet. They modeled a film-cooling configuration that included a plenum, jet tube, and cross-flow region above a flat plate. The jet injection angle was 35 degrees to the flat plate and the jet tube was cylindrical. The blowing ratio was varied from 0.5 to 2.0. The flowfield solution was obtained via the RANS method with the standard $k-\epsilon$ model on unstructured grids. They note that the counter-rotating vortex pair in the jet was the most significant mechanism affecting film-cooling performance. A key conclusion of this study was that the counter-rotating vortex pair in the jet originates from the boundary-layer vorticity emanating from the jet hole. In the companion papers, McGovern and Leylek [54] (part II) examined cylindrical jet holes with compound angle injection of the jet, Hyams and Leylek [9] (part III) examined shaped jet holes (also called shaped film-cooling) with streamwise injection of the jet, and Brittingham and Leylek [55] (part IV) examined shaped jet holes with compound angle injection of the jet.

A fundamental study by Tyagi and Acharya [30] numerically investigated a film-cooling flow using Large Eddy Simulation, where they used a dynamic mixed SGS model. They simulated a jet inclined at 35 degrees to the flat plate, where the jet tube was included in the domain. An immersed boundary method was used to satisfy the no-slip condition on the tube surface. They tested a Reynolds number of 11,100 at a blowing ratio of 0.5 and a Reynolds number of 22,200 at a blowing ratio of 1.0, where the Reynolds numbers

were based on the jet diameter and jet velocity. The counter-rotating vortex pair in the jet core, shear-layer vortices along the edge of the jet, and wake vortices downstream of the jet were visualized using components of the instantaneous streamwise vorticity, spanwise vorticity, and vertical vorticity, respectively. Shear-layer vortices along the downstream edge of the jet were found, but not along the upstream edge. A pair of counter-rotating wake vortices was observed just downstream of the jet; farther downstream of the jet, more wake vortices were observed and were distributed in the spanwise direction. A horseshoe vortex was not observed on the upstream edge of the jet. They visualized hairpin vortices using positive isosurfaces of the Laplacian of the pressure and postulated that the vortex structures mentioned above were all related to the hairpin dynamics.

Peet and Lele [32] performed a Large Eddy Simulation of a film-cooling flow using a computational domain that consisted of a plenum, jet tube, and cross-flow region above a flat plate. The jet tube was inclined at 35 degrees to the flat surface, the velocity ratio was 0.5, and the density ratio was 0.95. The turbulent cross-flow boundary layer was prescribed via a rescaling-recycling procedure, where the Reynolds number based on the momentum thickness was 938. A unique feature of this work was the simultaneous use of two codes for the simulations, where a low Mach number code was used for the plenum and jet tube and a compressible code was used for the flow above the flat plate. A dynamic Smagorinsky SGS model was used in both codes. It was observed that a separation bubble was created on the downstream wall of the jet tube, which was due to the flow making a sharp turn from the plenum into the tube. They also found that a "downstream spiral separation node (DSSN)" vortex was visible near the trailing-edge of the jet hole. This vortex was created by the entrainment of cross-flow fluid into the wake of the jet, owing to the low pressure region there. The DSSN vortex caused jet separation, resulting in a decrease in film-cooling effectiveness near the trailing edge of the jet.

A fundamental study by Muldoon and Acharya [33] investigated the behavior of the terms in the standard $k - \epsilon$ model for a film-cooling flow using DNS data. The study was

motivated by the need to improve the $k-\epsilon$ turbulence model, since the RANS approach is widely used for film-cooling flows. They calculated the terms in the exact k and ϵ equations and the standard $k-\epsilon$ model using DNS data, and then compared the exact and modeled terms. The velocity boundary condition at the jet exit was specified from time-averaged results from a separate precursor simulation that included the jet pipe and plenum. Due to the resolution requirements of DNS, they decided to exclude the plenum and jet pipe from the DNS and focus all resolution in the jet-crossflow interaction region above the flat plate.

They found that the expression for the eddy viscosity from the standard $k-\epsilon$ turbulence model ($C_\mu(k^2/\epsilon)Re$, where $C_\mu = 0.09$) over-predicted the eddy viscosity as compared to the DNS results. For the k-equation, they note that the eddy viscosity model yields an order of magnitude over-prediction in the production term and an order of magnitude under-prediction in the turbulent diffusion term. For the ϵ-equation, the standard $k-\epsilon$ model over-predicted the production term in the jet region and under-predicted the destruction term in the recirculation region behind the jet. For the turbulent diffusion term in the ϵ-equation, it was found that the eddy viscosity model under-predicted and over-predicted turbulent diffusion in the jet region and over-predicted turbulent diffusion in the horseshoe vortex region. They proposed two new damping functions that reduce the error in the eddy viscosity from the $k-\epsilon$ model.

There have been numerous studies that have investigated techniques to counteract the detrimental effects of vortex induction in flim-cooling flows. For example, Heidmann and Ekkad [18] presented an "antivortex" film-cooling hole design to improve film-cooling effectiveness. This design was studied numerically for a jet inclined at 30 degrees to a flat plate at a Reynolds number of 11,300 based on jet exit diameter and inlet conditions. They investigated a blowing ratio of 1.0 and density ratios of 1.05 and 2.0. The RANS approach was used with the $k-\omega$ turbulence model. This design included a main inclined jet tube with two additional jet tubes, where one tube was placed on each side of the main tube. The two additional side tubes were inclined in the spanwise direction and intersected the main tube.

The concept worked as follows: consider a row of main tubes, each with two side tubes. The coolant flow from adjacent side tubes from two adjacent main tubes interacts with each other to create a counter-rotating vortex pair. This vortex pair from the side tubes is of opposite rotation to the vortex pairs from the main tubes. The creation of the extra vortices from the side tubes is meant to help prevent jet lift-off and keep the coolant attached to the surface. The computational results showed that the antivortex design helped prevent lift-off and provided increased film-cooling effectiveness compared to the baseline case. In addition to the numerical study, experiments were performed by Dhungel et al. [24] to study the antivortex design and they also found that the design yielded improvements in film-cooling effectiveness.

Another technique that has been studied for improving film-cooling is pulsation of the coolant jet. Muldoon and Acharya [19] used DNS to investigate if pulsations of the coolant jet can improve film-cooling effectiveness. The jet was inclined at 35 degrees to the freestream, the Reynolds number based on the freestream velocity and jet exit diameter was 8000, and the Prandtl number was taken as unity. The duty cycle and Strouhal number were varied. They used a mesh with 737 (streamwise) x 193 (wall-normal) x 160 (spanwise) cells $\approx$ 23 million points. The velocity boundary condition at the jet exit was specified from time-averaged results from a separate precursor DNS that included the jet pipe and plenum. Modulation of this jet boundary condition was used to implement pulsation. They found that the pulsation created a "starting vortex" that becomes stretched in the streamwise direction due to the lower velocities near the trailing edge of the jet (region of separated flow) and higher velocities at the leading edge of the jet. This caused the vortex induction direction to be toward the wall, resulting in the jet attaching to the wall and thus leading to better cooling effectiveness. It was found that pulsation improved the cooling effectiveness in the separation region at the trailing edge of the jet, but that it eliminated the horseshoe vortices at the leading edge of the jet causing a local decrease in cooling effectiveness.

An experimental study of pulsing coolant jets in a film-cooling configuration was per-

formed by Coulthard et al. [20, 21]. They studied a film-cooling configuration with a single row of five cylindrical jets inclined at 35 degrees with respect to a flat plate. Solenoid valves were used to pulse the jets. The Reynolds number based on jet diameter and freestream velocity was 10,000. They examined blowing ratios of 0.25, 0.5, 1.0, and 1.5 and varied the pulsing frequencies and duty cycles. Velocity was measured using hot-wire anemometry and temperature was measured using cold-wire anemometry, thermocouples, and an infrared camera. It was found that low-frequency pulsation decreased effectiveness and that, in a few cases, high-frequency pulsation improved film-cooling effectiveness. In general, pulsation resulted in a decrease in film-cooling effectiveness, which was due to the coolant jet lifting off the surface during a portion of the pulsation cycle. The best film-cooling effectiveness was found using unpulsed jets at a blowing ratio of 0.5.

Placing a micro-ramp vortex generator downstream of a film-cooling hole has been studied as another way to improve cooling effectiveness. A micro-ramp in this configuration generates a pair of counter-rotating vortices that are rotating such that there is downwash between the vortices. These vortices can help entrain coolant from the jet and transport it to the surface to enhance cooling. If the vorticity field generated by the micro-ramp is strong enough, it can cancel the vorticity in the jet (since there is upwash between the counter-rotating vortices in the jet). This can prevent lift-off of the jet, resulting in enhanced cooling at the surface.

Rigby and Heidmann [22] numerically examined a micro-ramp downstream of a film-cooling jet. The jet was inclined at 30 degrees to the flat plate surface and the Reynolds number was 11,300 (based on freestream velocity and jet exit diameter). They used the Reynolds-Averaged Navier-Stokes (RANS) approach with a $k - \omega$ turbulence model with a mesh of approximately 1 million cells. They examined blowing ratios of 0.5, 1.0, 1.5, and 2.0. The addition of the micro-ramp helped cancel-out the vorticity in the counter-rotating vortex pair in the jet, leaving the pair of vortices generated by the micro-ramp. The micro-ramp also helped distribute the coolant over a wider area of the flat plate. Since the micro-ramp

vortices were generally closer to the surface and have a downwash component, they were able to transport coolant more effectively to the surface, which resulted in increased cooling effectiveness.

Zaman et al. [23] experimentally studied a micro-ramp placed downstream of a film-cooling jet at a Reynolds number of 11,400 based on the freestream velocity and jet exit diameter. The jet was inclined at an angle of 20 degrees to the freestream and the blowing ratio was $\sqrt{2}$. Hot-wire anemometry was used to measure mean velocity, r.m.s. velocity, and streamwise vorticity. A single hot-wire was used for measurements at the jet exit and two X-wires were used for measurements in streamwise planes downstream of the jet and micro-ramp. The vorticity generated by the micro-ramp canceled-out the vorticity in the jet, leaving only the counter-rotating vortices from the micro-ramp. As a result, the trajectory of the jet moved closer to the wall. They investigated varying the micro-ramp height, micro-ramp location, and micro-ramp edge curvature. It was observed that when the micro-ramp height is halved, the jet lifts off the wall and if the micro-ramp height is doubled, the jet is dissipated owing to an increase in turbulence intensities. They found that moving the micro-ramp over a distance of three diameters from the jet exit did not appreciably affect the results. Rounding-off the micro-ramp edges was found to weaken the counter-rotating vorticies leading to a decrease in effectiveness.

Na and Shih [56] numerically examined the effect of placing a ramp upstream of a film-cooling jet. They used the RANS approach with a $k - \epsilon$ turbulence model. Unlike the micro-ramp discussed previously, this ramp spanned the entire computational domain and was placed upstream of the jet. The ramp had an inclined surface on the windward side and a backward-facing step on the leeward side. The ramp redirected the cross-flow boundary layer to impact the jet flow above the surface, which prevented the formation of horseshoe vortices near the surface. They found that this allowed the jet to spread more in the spanwise direction, resulting in improved film-cooling effectiveness.

2.3 Immersed Boundary Methods

An immersed boundary method (IBM) is a numerical technique for incorporating the effect of a complex geometry into a computational fluid dynamics solution that uses a Cartesian mesh, where the mesh does not conform to the shape of the complex geometry. Put another way, a "boundary" is "immersed" within the Cartesian mesh. The Navier-Stokes equations are still solved using a standard Cartesian mesh, with extra steps taken to account for the presence of the immersed boundary. Thus immersed boundary methods can handle complex geometries while still retaining the advantages of using Cartesian meshes, such as ease of grid generation and straight-forward implementation of multigrid methods. In addition, the extra computational cost per time-step of incorporating the effect of the immersed boundary into the numerical solution of the governing equations is generally low. In this way, the IBM is more attractive than using boundary-fitted curvilinear meshes or unstructured meshes. Boundary-fitted curvilinear meshes require complex grid generation and require a coordinate transformation, which adds to the computational cost. Unstructured meshes also require complex grid generation and using multigrid methods becomes non-trivial. Another important issue to consider is moving boundaries. If using boundary-fitted curvilinear meshes or unstructured meshes, the mesh would have to be regenerated at each time-step to account for the new position of the boundary, resulting in a severe computational penalty. However, accounting for the movement of a boundary using an IBM is much easier since only the computations for the boundary itself change, while the mesh of the domain remains unchanged.

There are a number of techniques that are classified as an IBM, such as the forcing method, the ghost cell method, the immersed interface method, and the cut-cell method. The similarity between all these methods is that the Navier-Stokes equations are solved using a Cartesian mesh. The differences between the methods lie in how the immersed boundary is prescribed. The literature for these methods will be reviewed in the following paragraphs. It

should be noted that the literature on IBMs is vast, and this represents only a brief survey of these methods. For supplementary information, the reader is referred to reviews of immersed boundary methods given by Mittal and Iaccarino [57] and Iaccarino and Verzicco [58].

2.3.1 Forcing Methods

In the forcing method, a term (forcing function) is explicitly added to the governing equations that accounts for the presence of the immersed boundary in the mesh. This forcing function is applied to mesh points in the vicinity of the immersed boundary. The prescription of the forcing function varies from method to method, as we will see. The first IBM was proposed by Peskin [59, 60, 61], where he used a forcing method for the simulation of blood flow in the human heart, where the heart walls were represented as an immersed boundary and the flow was considered two-dimensional. Later, this work was extended to three-dimensional flow in the heart by McQueen and Peskin [62, 63]. The boundary was tracked via a Lagrangian method and the effect of the boundary was incorporated into the incompressible Navier-Stokes equations using a forcing term. The force was distributed to fluid nodes surrounding a given Lagrangian boundary point, where the distribution was governed by a discrete Dirac delta function. A disadvantage of the forcing method is that the boundary conditions on the immersed boundary are not exactly satisfied since the forces are distributed over a band of grid nodes.

Goldstein et al. [64] proposed an IBM where the forcing was given by an analytical expression that creates a feedback effect on the velocity field such that the prescribed boundary conditions are satisfied on the immersed boundary. The forcing expression contained two constants that needed to be tuned according to the frequencies of the flow. Their method was tested by simulating two-dimensional startup flow around a circular cylinder, three-dimensional channel flow, and ribbed channel flow. One disadvantage of this method is that to accurately enforce the boundary conditions implies that large values of the two constants are needed, which results in stability problems [57]. Another forcing method was presented

by Mohd-Yusof [65], where the forcing is derived from the discrete Navier-Stokes equations such that a prescribed velocity will be achieved on the immersed boundary. Linear interpolation was used to compute velocities at grid points located nearest to (and inside) the immersed boundary using the nearest fluid velocity points and prescribed boundary velocities. This velocity interpolant is then used in the forcing calculation. This technique has a number of advantages, including the fact that the boundary conditions on the immersed boundary are exactly satisfied and there are no free parameters to choose (unlike the work of Goldstein et al. [64] mentioned above where there are two parameters to choose). To test this method, it was applied to the simulation of laminar flow through a three-dimensional ribbed channel.

Fadlun et al. [66] compared the forcing methods of Goldstein et al. [64] and Mohd-Yusof [65] to assess the efficiency and accuracy of the two methods. It was found that both methods yield similar results, but the method in [65] was found to be more efficient. The method of [65] was used to simulate the vortex ring formation from a curvilinear nozzle, flow around a sphere, and turbulent flow inside an internal combustion cylinder with a moving piston. The latter simulation was the first reported use of an immersed boundary method combined with an LES SGS model, and excellent agreement with experiment was obtained.

Choi et al. [67] proposed an IBM based on forcing that was implemented for the three-dimensional incompressible Navier-Stokes equations. The immersed boundary was represented using a cloud of points that could be arranged in a structured or unstructured manner. The velocities near the immersed boundary were calculated using interpolations of tangential and normal components to the boundary. They used a power law function to prescribe the tangential velocity near the surface, and they note that the power law allows their IBM to approximate the energizing effects of a turbulent boundary layer at high Reynolds number. Their IBM was tested for a variety of problems, including flow over a circular cylinder, an in-line oscillating cylinder, a sphere, a NACA 0012 airfoil, and a mannequin. Good agreement was found compared with previous results. Ghosh et al. [68] extended the

IBM of Choi et al. [67] to handle compressible, turbulent flows using a RANS approach and a hybrid LES/RANS approach. This work was used to simulate the flow past micro-ramp vortex generators for controlling oblique shock wave/boundary layer interactions.

Gao et al. [69] presented a forcing method that is an extension of the work in [65] and [66]. The grid points closest to (and inside of) the immersed surface were selected as forcing points. A novel feature of their method is that a second-order Taylor series expansion was used to compute values at the ghost point locations. This has two primary advantages. First, this method eliminates the need to perform matrix inversions to obtain the ghost point values, which avoids the possibility of trying to invert an ill-conditoned matrix that occurs when the immersed boundary is close to a fluid point. Second, it avoids the possibility of numerical divergence when the absolute value at the ghost point is much larger than the neighboring fluid points, which can happen if the boundary point is close to one of the fluid nodes used in the extrapolation. To demonstrate their method, they simulated the flow past a circular cylinder, a sphere, two cylinders that were side-by-side, and an array of 18 staggered cylinders.

Kim et al. [70] proposed a forcing method based on that of Mohd-Yusof [65] and used the finite-volume approach. A unique feature of their method is that they applied a mass source or sink in the cell containing the immersed boundary to satisfy conservation of mass. They tested this method for decaying vortices, flow past a cylinder, and flow past a sphere with good agreement versus previous numerical and experimental results. This method was extended to moving boundary problems by Kim and Choi [71], where they considered simulations of inline and transverse oscillation of a circular cylinder, vortex-induced vibration of a circular cylinder, and a freely falling object under gravity.

Tyagi and Acharya [72] developed a forcing method for Large Eddy Simulation. They proposed using forcing on both sides of the immersed boundary in order to avoid ill-conditioned weights that can be produced using one-sided forcing (forcing only on the inside of the immersed boundary). Their two-sided forcing strategy was shown to work well with moving

immersed boundaries. Their IBM was used to simulate the laminar flow past a heated cylinder in a channel and turbulent flow in a film-cooling configuration, where good agreement was found compared with measurements. They also applied their method to study turbulent mixing inside a trapped vortex combustor and a moving stator-rotor interaction.

2.3.2 Ghost Cell Methods

The ghost-cell methods account for the presence of the boundary implicitly, meaning no forcing function is added to the governing equations. Instead, ghost cells located inside the obstacle are adjusted such that desired boundary conditions are satisfied on the immersed boundary. Ghost cells are defined as computational cells inside the obstacle and near the boundary, where at least one neighbor is a fluid-cell. This method is relatively straightforward to implement, has a low computational cost, and by design exactly satisfies the desired boundary conditions along the immersed boundary.

Tseng and Ferziger [73] presented a second-order accurate ghost cell method used with the Navier-Stokes equations. The ghost cell velocity values were computed as a linear interpolation of the boundary condition values at the immersed boundary and an image (or mirror) point in the flow. The mirror point values were found using a linear interpolation using the neighboring fluid points and the boundary point. Linear interpolation was also used for the pressure boundary condition (zero normal gradient) at the immersed surface. This interpolation caused a mass flux to be generated across the immersed boundary. Despite this, good results were obtained using this method. They performed simulations of flow past a circular cylinder and an LES of turbulent flow over a wavy surface to demonstrate the method.

Mark and van Wachem [74] developed a second-order accurate ghost cell method for three-dimensional flows. The immersed boundary was represented using triangular elements. They proposed and compared two methods: a vertex-constraining method and a mirroring method. In the vertex-constraining method, vertices in the triangulation that are closest to

the ghost cell pressure points are constrained to a desired velocity (these points were defined as "control points"). The ghost cell velocity points inside the immersed boundary are set such that a trilinear interpolation of neighboring velocities onto the control points yielded the desired velocity at the immersed boundary. In the mirroring method, for each ghost velocity point a mirror point is placed in the fluid such that the boundary is equidistant from this point and the corresponding ghost point. Trilinear interpolation was used on the mirror point using neighboring fluid points, and the ghost velocity is then set such that an interpolation between the ghost point and mirror point yields a desired velocity exactly at the boundary. In both methods, the velocities of the ghost cells were excluded from the pressure equation to ensure no mass flux was generated across the immersed boundary. The methods were compared by simulating the flow over a sphere and it was found that the mirroring method was more stable and had a faster convergence rate. Lastly, they use the mirroring method to simulate the flow around a cluster of non-spherical obstacles of various sizes.

At the same time as [74], Mittal et al. [75] presented a ghost cell method for three-dimensional incompressible flows. Their method was capable of handling stationary, moving, or deforming obstacles. The immersed boundaries were represented using triangular elements. The procedure for setting the ghost cell values is similar to the mirroring method of [74], except that for the pressure boundary condition a second-order central-difference was used across the boundary using the ghost cell pressure and the mirror pressure (obtained from interpolation of surrounding pressures). The method was used to study flow past a circular cylinder, a NACA 0008 airfoil, and a sphere. In addition, they examined flow past suddenly accelerated bodies including a suddenly accelerated normal plate and suddenly accelerated circular cylinder. They also investigated flow past complex moving bodies including a pectoral fin of a fish and the flight of a dragonfly.

Shinn et al. [76] developed a ghost cell method for two-dimensional, incompressible flows. This method used a mirroring technique and solved the pressure equation in the ghost cells

to ensure that both local and global mass conservation was obtained, similar to that of [74]. The authors noted that using a ghost point velocity in the interpolation procedure for the mirror point can lead to numerical divergence if the ghost point is close to the boundary. To remedy this, the ghost value was removed from the interpolation by using the relation between the ghost point, boundary point, and mirror point. Their ghost cell method was tested by simulating shear-driven flows (including a triangular cavity with moving walls and sinusoidal cavities with a moving wall) and buoyancy-driven flows (including natural convection in square, triangular, and circular enclosures).

In addition to problems with a solid immersed boundary, the ghost cell method has also been used for multiphase flows. For example, Fedkiw et al. [77] used a ghost cell method (called the "ghost fluid method") with the two-dimensional compressible Euler equations to study two-phase flow. They used a level set function to keep track of the interface between the two fluids and the ghost fluid method was used to maintain a sharp interface. Fedkiw et al. [78] extended the ghost fluid method to handle deflagration and detonation discontinuities.

2.3.3 Immersed Interface Methods

In the immersed interface method, jump conditions are derived that account for the boundary and are used in conjunction with the governing equations. This method provides a sharp representation of the boundary and is superior to some forcing methods that yield a diffuse representation due to applying the force across a band of mesh points.

Early work on the immersed interface method was presented by LeVeque and Li [79]. This was later used for the simulation of Stokes flow problems by LeVeque and Li [80]. Li and Lai [81] proposed an immersed interface method for the two-dimensional incompressible Navier-Stokes equations with singular forces along an obstacle boundary. Their method was based on a second-order projection method, which was modified by using correction terms only at grid points near or on the interface. They derived interface relations for determining

the correction terms to the original projection method. They found that the solution was second-order accurate for the velocity and nearly second-order accurate for the pressure (including grid points near or on the interface).

Lee and LeVeque [82] present an immersed interface method for the two-dimensional incompressible Navier-Stokes equations. They solved the equations numerically using a projection method. Jump conditions were imposed for pressure and velocity, where the jump in pressure was imposed while solving the pressure-Poisson equation. The immersed interface was tracked using marker points that move with the fluid. Their approach applied the tangential component of the force at the interface by spreading it using discrete delta functions. If the force was purely in the normal direction, then discrete delta functions were not used. They note that, as compared to their own approach, the approach given by Li and Lai [81] may yield better accuracy, but is more difficult to implement. They presented example problems which involved a pressurized circular membrane that was allowed to deform. Second-order accuracy was achieved for the test cases they examined. They found that compared with a forcing-based immersed boundary method, the pressure jump was captured more sharply using the immersed interface method.

Xu and Wang [83] derived jump conditions for the immersed interface method for the three-dimensional incompressible Navier-Stokes equations. They derived the jump conditions of all first-order, second-order, and third-order spatial derivatives of the velocity and pressure. The jump conditions of first-order and second-order time derivatives of velocity were also derived. Xu and Wang [84] extended this work and presented the implementation and results for an immersed interface method for three-dimensional fluid flows. Spatial discretization was carried out using the MAC scheme and time integration was performed using the 4th-order Runge-Kutta scheme. They used the jump conditions that they had previously derived in [83]. Their method had near second-order accuracy for velocity and between first and second order accuracy for the pressure. They also found that their method conserves the volume enclosed by a no-penetration boundary. Parametric triangulation of the interface

was used to locate the intersections and to interpolate jump conditions from the Lagrangian markers to the intersections. They tested their method for flow inside a rotating object, flow induced by a relaxing balloon, flow past a stationary sphere, flow around a hovering flapper, and flow induced by multiple spinning spheres.

Karagiozis et al. [85] presented an immersed interface method for the compressible Navier-Stokes equations. Immersed interfaces were represented in the numerical solution by modifying the discretization stencils for irregular grid nodes near the boundary. Their method did not require jump conditions at the interface, which is a notable difference compared to other immersed interface implementations. Their method possessed low numerical dissipation and they used summation-by-parts operators to construct approximations of first-order derivatives that were stable at irregular grid points. The method was tested for a rotating circular cylinder, flow over one, two, and three circular cylinders along with flow past two star-shaped bodies and flow past a thickened parabolic section. Additionally, scattering of a sound wave by a circular cylinder was simulated. Good agreement versus previous experimental and numerical data was observed for the flow over a circular cylinder.

2.3.4 Cut-Cell Methods

The cut-cell method is a finite-volume-based approach where the intersection of the immersed boundary with the mesh cells is used to create new cells by "cutting" the original cells along the immersed boundary lines. Thus, square cells become trapezoidal cells (in two-dimensions) and cubical cells become polyhedrons (in three-dimensions) near the immersed boundary. The cut-cell method has the advantage of satisfying local and global conservation by design, but has the disadvantage of increased programming complexity.

The cut-cell method was originally proposed by Clarke et al. [86], where they used the technique for simulation of inviscid flow past multi-element airfoils. Ye et al. [87] presented a two-dimensional cut-cell finite-volume-based method for viscous, incompressible flows. They simulated flow past a circular cylinder placed near a moving wall in Stokes

flow, a circular cylinder, a random array of cylinders, and a cascade of airfoils. Kirkpatrick et al. [88] developed a three-dimensional cut-cell method for the incompressible Navier-Stokes equations, where the immersed boundaries were prescribed using quadric surfaces. They demonstrated that the cut-cell method yields significant improvement compared with a stair-step boundary. Their method was tested for flow through a skewed channel, flow past a cylinder, and flow past a hemispherical bump on a flat plate. Recent work by Hartmann et al. [89] was the first work to present a three-dimensional cut-cell implementation for compressible, viscous flows. Their implementation was capable of using adaptive meshes. They presented the method for flow past a circular cylinder and flow past a sphere. The cut-cell method has been used for simulating a number of other interesting problems, such as flapping wings for biomimetic flight [90] and synthetic jets driven by a diaphragm [91].

Chapter 3

Governing Equations and Numerical Methodology

3.1 Governing Equations

The governing equations to be solved in the present work are the conservation of mass, momentum, and energy for an incompressible flow. In dimensional form the equations are

$$\nabla \cdot \mathbf{u} = 0 \tag{3.1}$$

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) \right) = -\nabla p + \nabla \cdot (\mu \nabla \mathbf{u}) + \mathbf{F} \tag{3.2}$$

$$\rho c_p \left(\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{u}T) \right) = \nabla \cdot (k \nabla T) \tag{3.3}$$

where $\mathbf{u}$ is the velocity vector with Cartesian components (u, v, w), ρ is the density, t is the time, p is the pressure, μ is the dynamic viscosity of the fluid, $\mathbf{F}$ is a body source term, T is the temperature, k is the thermal conductivity of the fluid, and c_p is the specific heat at constant pressure. Note that the temperature is one-way coupled with the velocity field and is thus treated as a passive conserved scalar.

The governing equations can be non-dimensionalized using

$$x^* = \frac{x}{L} \qquad y^* = \frac{y}{L} \qquad z^* = \frac{z}{L} \qquad t^* = \frac{t}{(L/U)}$$

$$\mathbf{u}^* = \frac{\mathbf{u}}{U} \qquad p^* = \frac{p}{\rho U^2} \qquad T^* = \frac{(T - T_1)}{(T_2 - T_1)} \qquad \nabla^* = L\nabla \tag{3.4}$$

where L is the characteristic length, U is the characteristic velocity, and T_1 and T_2 are

characteristic reference temperatures. These characteristic scales will be identified for each problem. Substituting the non-dimensionalizations of Equation (3.4) in Equations (3.1) to (3.3) yields the non-dimensional form of the governing equations:

$$\nabla^* \cdot \mathbf{u}^* = 0 \tag{3.5}$$

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + \nabla^* \cdot (\mathbf{u}^* \mathbf{u}^*) = -\nabla^* p^* + \nabla^* \cdot \left(\frac{1}{Re} \nabla^* \mathbf{u}^* \right) + \mathbf{F}^* \tag{3.6}$$

$$\frac{\partial T^*}{\partial t^*} + \nabla^* \cdot (\mathbf{u}^* T^*) = \nabla^* \cdot \left(\frac{1}{RePr} \nabla^* T^* \right) \tag{3.7}$$

where the Reynolds number is $Re = \rho UL/\mu$ and Prandtl number is $Pr = \mu c_p/k$. Note that we have assumed that the diffusion coefficient of the momentum and energy equations may vary spatially. This general form allows the incorporation of an eddy viscosity for LES, which will cause the effective diffusion coefficient to vary. For DNS, the effective diffusion is constant.

3.2 Governing Equations for Large-Eddy Simulation

3.2.1 Derivation of the Filtered Equations

In Large Eddy Simulation (LES), the governing equations are solved numerically such that the larger, energy-containing turbulent structures are resolved, while the smaller scales are modeled. To derive the governing equations for LES, we apply a filter to the governing equations, which will decompose the velocity field $\mathbf{u}$ into a filtered velocity $\widetilde{\mathbf{u}}$ and a residual velocity $\mathbf{u}'$:

$$\mathbf{u} = \widetilde{\mathbf{u}} + \mathbf{u}' \tag{3.8}$$

The filtering operation is applied using an integral over the entire domain (defined as Ω),

$$\widetilde{\mathbf{u}}(\mathbf{x}) = \int_{\Omega} G(\mathbf{x}, \mathbf{x}')\mathbf{u}(\mathbf{x}')\ dx'_1 dx'_2 dx'_3 \tag{3.9}$$

where G is the filter function and $\mathbf{x}, \mathbf{x}'$ are position vectors. Since we are using the Finite Volume Method, the filtering is implicit, where the mesh acts as the filter. This is the same as using a box filter as the filter function, given as

$$G(\mathbf{x}, \mathbf{x}') = \begin{cases} 1/\Delta\Omega_{cell} & \text{if } \mathbf{x} \text{ and } \mathbf{x}' \text{ are in the same computational cell} \\ 0 & \text{if } \mathbf{x} \text{ and } \mathbf{x}' \text{ are in different computational cells} \end{cases} \tag{3.10}$$

where $\Delta\Omega_{cell}$ is the volume of a cell in the mesh [92]. Before applying the filter, let us write the dimensional governing equations, Equations (3.1) to (3.3), in indicial notation, which is helpful in the present context. These can be written as

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{3.11}$$

$$\rho\left(\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j)\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\frac{\partial u_i}{\partial x_j}\right) + F_i \tag{3.12}$$

$$\rho c_p\left(\frac{\partial T}{\partial t} + \frac{\partial}{\partial x_j}(u_j T)\right) = \frac{\partial}{\partial x_j}\left(k\frac{\partial T}{\partial x_j}\right) \tag{3.13}$$

where summation over a repeated index is implied. Applying the filter to the governing equations produces

$$\frac{\partial \widetilde{u}_i}{\partial x_i} = 0 \tag{3.14}$$

$$\rho\left(\frac{\partial \widetilde{u}_i}{\partial t} + \frac{\partial}{\partial x_j}(\widetilde{u_i u_j})\right) = -\frac{\partial \widetilde{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\frac{\partial \widetilde{u}_i}{\partial x_j}\right) + F_i \tag{3.15}$$

$$\rho c_p\left(\frac{\partial \widetilde{T}}{\partial t} + \frac{\partial}{\partial x_j}(\widetilde{u_j T})\right) = \frac{\partial}{\partial x_j}\left(k\frac{\partial \widetilde{T}}{\partial x_j}\right) \tag{3.16}$$

Now we define the sub-grid-scale stress tensor as

$$\tau_{ij} = \widetilde{u_i u_j} - \widetilde{u}_i \widetilde{u}_j \tag{3.17}$$

and substitute it into the filtered momentum equation above which gives

$$\rho \left(\frac{\partial \widetilde{u}_i}{\partial t} + \frac{\partial}{\partial x_j}(\widetilde{u}_i \widetilde{u}_j) \right) = -\frac{\partial \widetilde{p}}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\mu \frac{\partial \widetilde{u}_i}{\partial x_j} \right) + F_i - \rho \frac{\partial \tau_{ij}}{\partial x_j} \tag{3.18}$$

The last term must be modeled, which we do using a linear eddy viscosity model of the form

$$\tau_{ij} - (1/3)\delta_{ij}\tau_{kk} = -2\nu_t \widetilde{S_{ij}} \tag{3.19}$$

where ν_t is the turbulent viscosity and $\widetilde{S_{ij}}$ is the filtered rate-of-strain

$$\widetilde{S}_{ij} = \frac{1}{2} \left(\frac{\partial \widetilde{u}_i}{\partial x_j} + \frac{\partial \widetilde{u}_j}{\partial x_i} \right) \tag{3.20}$$

Substituting the eddy viscosity model of Equation (3.19) into Equation (3.18) and using a modified pressure defined as $\widehat{\widetilde{p}} \equiv \widetilde{p} + (1/3)\rho\tau_{kk}$ yields

$$\rho \left(\frac{\partial \widetilde{u}_i}{\partial t} + \frac{\partial}{\partial x_j}(\widetilde{u}_i \widetilde{u}_j) \right) = -\frac{\partial \widehat{\widetilde{p}}}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\mu \frac{\partial \widetilde{u}_i}{\partial x_j} \right) + F_i + \frac{\partial}{\partial x_j} \left(\mu_t \left(\frac{\partial \widetilde{u}_i}{\partial x_j} + \frac{\partial \widetilde{u}_j}{\partial x_i} \right) \right) \tag{3.21}$$

Using the incompressibility constraint and neglecting the non-uniform eddy viscosity term[1] gives the final form of the filtered momentum equation as

$$\rho \left(\frac{\partial \widetilde{u}_i}{\partial t} + \frac{\partial}{\partial x_j}(\widetilde{u}_i \widetilde{u}_j) \right) = -\frac{\partial \widehat{\widetilde{p}}}{\partial x_i} + \frac{\partial}{\partial x_j} \left((\mu + \mu_t) \frac{\partial \widetilde{u}_i}{\partial x_j} \right) + F_i \tag{3.22}$$

Turning our attention to the filtered energy equation, we define the sub-grid scale heat

[1] This term is usually small and hence can be neglected.

flux vector q_i as

$$q_j = \widetilde{u_j T} - \widetilde{u}_j \widetilde{T} \tag{3.23}$$

and substituting into the filtered energy equation we find

$$\rho c_p \left(\frac{\partial \widetilde{T}}{\partial t} + \frac{\partial}{\partial x_j}(\widetilde{u}_j \widetilde{T}) \right) = \frac{\partial}{\partial x_j} \left(k \frac{\partial \widetilde{T}}{\partial x_j} \right) - \rho c_p \frac{\partial q_j}{\partial x_j} \tag{3.24}$$

The last term must be modeled, which we do using a linear eddy diffusivity model of the form [93]

$$q_j = -\frac{\nu_t}{Pr_t} \frac{\partial \widetilde{T}}{\partial x_j} \tag{3.25}$$

where Pr_t is the turbulent Prandtl number, and $Pr_t = 0.9$ for air [94]. Substituting this model into Equation (3.24) and noting that the turbulent thermal conductivity is $k_t = c_p(\mu_t / Pr_t)$ yields the final result

$$\rho c_p \left(\frac{\partial \widetilde{T}}{\partial t} + \frac{\partial}{\partial x_j}(\widetilde{u}_j \widetilde{T}) \right) = \frac{\partial}{\partial x_j} \left((k + k_t) \frac{\partial \widetilde{T}}{\partial x_j} \right) \tag{3.26}$$

3.2.2 Models for the Eddy Viscosity

To complete the LES formulation, we need to specify the models for the eddy viscosity. In the present work we use two models for the eddy viscosity: the Smagorinsky model and the WALE model (Wall-Adapting Local Eddy-viscosity).

The Smagorinsky Model

In the Smagorinsky model [95] the eddy viscosity is given by

$$\nu_t = (C_s \Delta)^2 \widetilde{S} \tag{3.27}$$

where C_s is the Smagorinsky coefficient and is taken as a constant (typically $C_s = 0.1$), Δ is the filter width, and $\widetilde{S}$ is the filtered rate-of-strain. The filter width for a given cell is

$$\Delta = (\Delta x_i \Delta y_j \Delta z_k)^{1/3} \tag{3.28}$$

and the filtered rate-of-strain is

$$\widetilde{S} = (2\widetilde{S}_{ij}\widetilde{S}_{ij})^{1/2} \tag{3.29}$$

This model is simple to implement and is computationally efficient, but it does have some drawbacks. For example, in laminar flows or near a wall the eddy viscosity should go to zero, but the Smagorinsky model produces non-zero eddy viscosity for these cases. Wall damping can be used to correct the near-wall behavior, but the near-wall scaling of the eddy viscosity could still be incorrect (for example, using Van Driest damping the near-wall scaling of the eddy viscosity is $O(y^2)$, whereas it should be $O(y^3)$).

The WALE Model

A better eddy viscosity model is the WALE model, proposed by Nicoud and Ducros [96]. This model produces an eddy viscosity that goes to zero near a wall without a dynamic procedure or wall damping, and the eddy viscosity exhibits the correct $O(y^3)$ scaling near a wall. In addition, it is relatively easy to implement. The WALE model for the eddy viscosity is given by

$$\nu_t = (C_w \Delta)^2 \frac{(S^d_{ij} S^d_{ij})^{3/2}}{(\widetilde{S}_{ij}\widetilde{S}_{ij})^{5/2} + (S^d_{ij} S^d_{ij})^{5/4}} \tag{3.30}$$

where C_w is the WALE constant (with a typical range of $0.55 \leq C_w \leq 0.60$ for a variety of flows). The tensor S^d_{ij} is defined as

$$S^d_{ij} = \frac{1}{2}(\widetilde{g}^2_{ij} + \widetilde{g}^2_{ji}) - \frac{1}{3}\delta_{ij}\widetilde{g}^2_{kk} \tag{3.31}$$

where

$$\widetilde{g}_{ij}^2 = \widetilde{g}_{ik}\widetilde{g}_{kj} \tag{3.32}$$

and

$$\widetilde{g}_{ij} = \frac{\partial \widetilde{u}_i}{\partial x_j} \tag{3.33}$$

3.2.3 Summary

Let us summarize the filtered governing equations for LES as derived previously,

$$\nabla \cdot \widetilde{\mathbf{u}} = 0 \tag{3.34}$$

$$\rho\left(\frac{\partial \widetilde{\mathbf{u}}}{\partial t} + \nabla \cdot (\widetilde{\mathbf{u}}\widetilde{\mathbf{u}})\right) = -\nabla\widehat{\widetilde{p}} + \nabla \cdot ((\mu + \mu_t)\nabla\widetilde{\mathbf{u}}) + \mathbf{F} \tag{3.35}$$

$$\rho c_p\left(\frac{\partial \widetilde{T}}{\partial t} + \nabla \cdot (\widetilde{\mathbf{u}}\widetilde{T})\right) = \nabla \cdot ((k + k_t)\nabla\widetilde{T}) \tag{3.36}$$

or non-dimensionally

$$\nabla^* \cdot \widetilde{\mathbf{u}}^* = 0 \tag{3.37}$$

$$\frac{\partial \widetilde{\mathbf{u}}^*}{\partial t^*} + \nabla^* \cdot (\widetilde{\mathbf{u}}^*\widetilde{\mathbf{u}}^*) = -\nabla^*\widehat{\widetilde{p}^*} + \nabla^* \cdot \left(\left(\frac{1}{Re} + \frac{1}{Re_t}\right)\nabla^*\widetilde{\mathbf{u}}^*\right) + \mathbf{F}^* \tag{3.38}$$

$$\frac{\partial \widetilde{T}^*}{\partial t^*} + \nabla^* \cdot (\widetilde{\mathbf{u}}^*\widetilde{T}^*) = \nabla^* \cdot \left(\left(\frac{1}{RePr} + \frac{1}{Re_tPr_t}\right)\nabla^*\widetilde{T}^*\right) \tag{3.39}$$

Comparing these LES equations to the DNS equations (Equations (3.1) to (3.3) and Equations (3.5) to (3.7)), we observe that they are of the same form, except that the effective viscosity and effective conductivity have contributions from SGS models for LES. In the next section, the details of the numerical methods used to solve the governing equations are given.

3.3 Numerical Solution of the Governing Equations

3.3.1 Overview

The governing equations are solved numerically by discretizing them temporally and spatially. The equations are solved using the fractional-step method [97, 98, 99] and spatial discretization was performed using the finite volume method with a Cartesian mesh. In the fractional step method, the momentum equation is first solved without the pressure gradient to obtain a provisional velocity field $\widehat{\mathbf{u}}$ that is not divergence-free (thus mass conservation is not satisfied). A pressure-Poisson equation (PPE) is then solved to obtain the updated pressure field p^{n+1}. This pressure field is used to enforce conservation of mass by correcting the provisional velocity to obtain the updated velocity field, $\mathbf{u}^{n+1}$, which is divergence-free. The following discretization is derived starting from the dimensional form of the governing equations, Equations (3.1) to (3.3). The resulting discrete equations can be used for simulations of unsteady laminar flow, and for DNS and LES of turbulent flows. In the case of laminar flow and DNS, the effective diffusion coefficient for the momentum and energy equations is constant, but for LES it is variable due to the addition of an eddy viscosity. Also note that the resulting discretization can be used to solve the non-dimensional form of the equations by simply substituting the following as input parameters: $\rho := 1$, $c_p := 1$, $\mu := \frac{1}{Re}$, and $k := \frac{1}{RePr}$. Details on how to implement the flow solver on a GPU will be given, and a 3D ghost cell method will be derived and discussed, which is used to handle flow past complex geometries.

3.3.2 Fractional Step Method

We start by writing the momentum equation, Equation (3.2), with only the time derivative on the left-hand-side

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p - \rho\nabla\cdot(\mathbf{uu}) + \nabla\cdot(\mu\nabla\mathbf{u}) + \mathbf{F} \tag{3.40}$$

and combine the convection and diffusion terms into a single vector $\mathbf{H}$ as

$$\mathbf{H} \equiv -\rho\nabla\cdot(\mathbf{uu}) + \nabla\cdot(\mu\nabla\mathbf{u}) \tag{3.41}$$

which has components (H_u, H_v, H_w). Thus, we have

$$\rho\frac{\partial\mathbf{u}}{\partial t} = -\nabla p + \mathbf{H} + \mathbf{F} \tag{3.42}$$

Writing this equation discretely in time using the explicit second-order accurate Adams-Bashforth method for the convection and diffusion terms yields

$$\rho\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} = -\nabla p^{n+1} + \frac{3}{2}\mathbf{H}^n - \frac{1}{2}\mathbf{H}^{n-1} + \mathbf{F}^n \tag{3.43}$$

We now perform a time-splitting on Equation (3.43), which gives

$$\rho\frac{\widehat{\mathbf{u}}-\mathbf{u}^n}{\Delta t} = \frac{3}{2}\mathbf{H}^n - \frac{1}{2}\mathbf{H}^{n-1} + \mathbf{F}^n \tag{3.44}$$

$$\rho\frac{\mathbf{u}^{n+1}-\widehat{\mathbf{u}}}{\Delta t} = -\nabla p^{n+1} \tag{3.45}$$

Equation (3.44) advances the velocity field to a provisional state $\widehat{\mathbf{u}}$ that is not divergence-free (thus mass conservation is not satisfied). Then Equation (3.45) corrects this provisional state such that mass conservation is satisfied using the updated pressure field p^{n+1}. The equation governing the pressure can be derived by taking the divergence, $\nabla\cdot(\ )$, of the discrete momentum equation, Equation (3.45), and using the discrete continuity equation, $\nabla\cdot\mathbf{u}^{n+1} = 0$. This produces the pressure-Poisson equation:

$$\nabla^2 p^{n+1} = \frac{\rho}{\Delta t}(\nabla\cdot\widehat{\mathbf{u}}) \tag{3.46}$$

Equations (3.44) to (3.46) represent the fractional step method for solving the incompressible Navier-Stokes equations, which is summarized in Algorithm 1.

Algorithm 1: Fractional Step Method

for $n = 1$ **to** *total number of time-steps* **do**

$\widehat{\mathbf{u}} \leftarrow \rho\frac{\widehat{\mathbf{u}}-\mathbf{u}^n}{\Delta t} = \frac{3}{2}\mathbf{H}^n - \frac{1}{2}\mathbf{H}^{n-1} + \mathbf{F}^n$

$p^{n+1} \leftarrow \nabla^2 p^{n+1} = \frac{\rho}{\Delta t}(\nabla \cdot \widehat{\mathbf{u}})$

$\mathbf{u}^{n+1} \leftarrow \rho\frac{\mathbf{u}^{n+1}-\widehat{\mathbf{u}}}{\Delta t} = -\nabla p^{n+1}$

end

3.3.3 Spatial Discretization

Now that the governing equations have been discretized temporally, the next step is to discretize the equations spatially, which is done using the Finite Volume Method. We first divide the flow domain into cells to create a mesh of the domain. The variables are stored in a staggered arrangement, which helps conserve energy and also avoids the development of unphysical pressures. The Cartesian velocity components (u, v, w) are stored at cell faces, and the scalars ($\phi = p, T, \rho$, and μ) are stored at the cell center, as shown in Figure 3.1. The mesh spacings for a given cell at location (i, j, k) are Δx_i, Δy_j, Δz_k, where the spacings can be non-uniform.

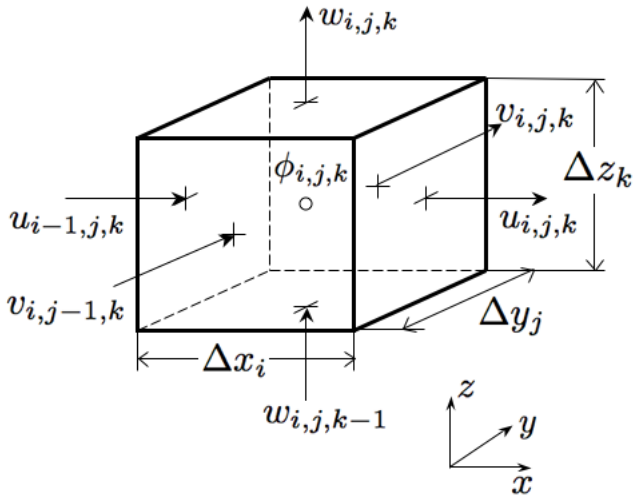


Figure 3.1: Cartesian mesh cell at location i, j, k using staggered arrangement of variables.

In the Finite Volume Method, the equations are integrated across a particular domain. The domain is taken as a control volume (CV), and each of the variables we wish to solve for (u, v, w, p, T) lie at the center of their own control volume. For scalars the control volume is coincident with the grid cell. For the velocity components, the control volume is offset from the grid cell by half a grid cell length in the positive coordinate direction (for example, the u-CV is offset by half a grid cell length in the positive x-direction). To illustrate, the scalar-CV and the u-CV are shown in Figures 3.2 and 3.3, respectively. The control volumes are indicated in gray with a dashed outline, and the solid black lines indicate the cell lines. The six faces of a control volume are referred to as the x_+, x_-, y_+, y_-, z_+, and z_- faces, which will be used in subsequent derivations.

Spatial Discretization of Momentum Equation for $\widehat{\mathbf{u}}$

The spatial discretization of the time-split momentum equations, Equations (3.44) and (3.45), will be derived for the u-component only, since the v- and w-components are analogous. The u-component from Equation (3.44) is

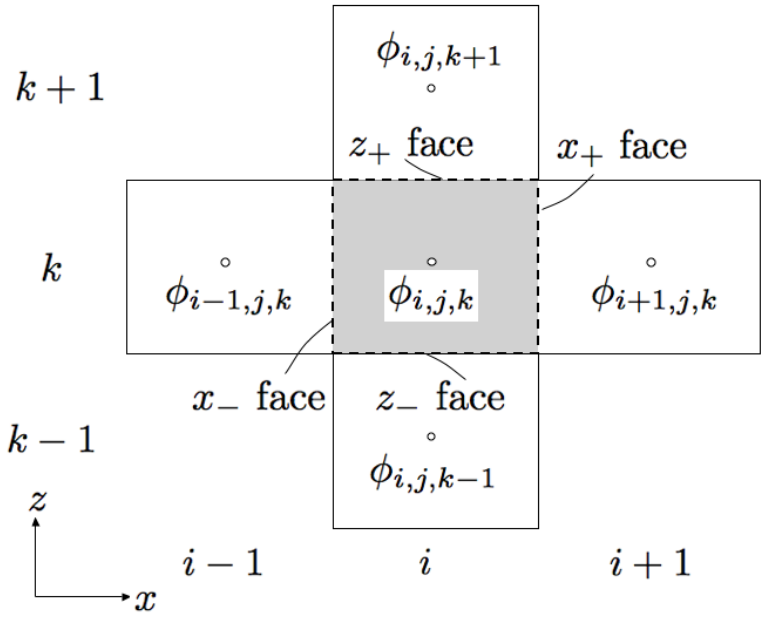
$$\rho\frac{\widehat{u} - u^n}{\Delta t} = \frac{3}{2}H_u^n - \frac{1}{2}H_u^{n-1} + F_u^n \tag{3.47}$$

Solving this equation for $\widehat{u}$ gives

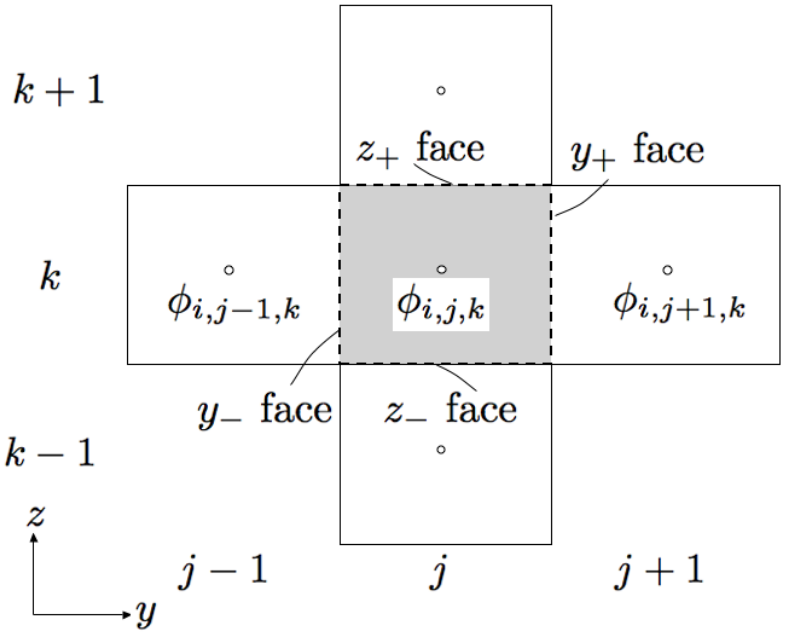
$$\widehat{u} = u^n + \frac{\Delta t}{\rho}\left(\frac{3}{2}H_u^n - \frac{1}{2}H_u^{n-1} + F_u^n\right) \tag{3.48}$$

We now apply the Finite Volume Method to Equation (3.48) by integrating over the u-control volume (defined as a region Ω)

$$\int_\Omega \widehat{u}\; d\Omega = \int_\Omega u^n\; d\Omega + \frac{\Delta t}{\rho}\left(\frac{3}{2}\int_\Omega H_u^n\; d\Omega - \frac{1}{2}\int_\Omega H_u^{n-1}\; d\Omega + \int_\Omega F_u\; d\Omega\right) \tag{3.49}$$

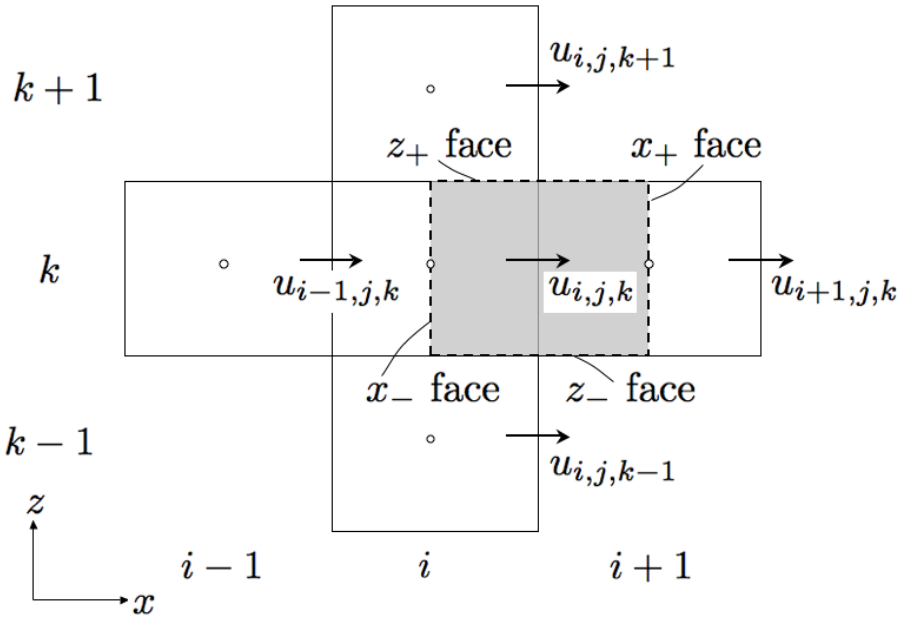


(a) $x-z$ view of scalar-CV

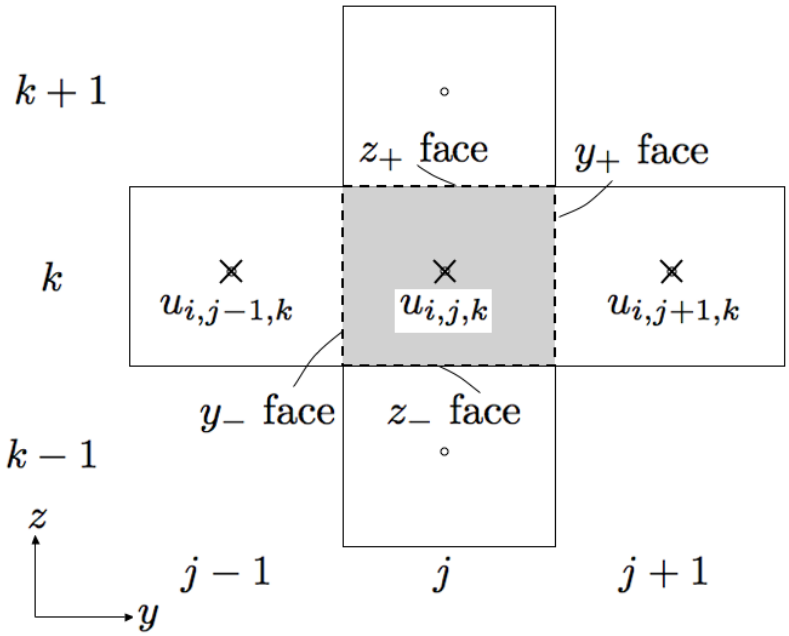


(b) $y-z$ view of scalar-CV

Figure 3.2: The scalar control volume (scalar-CV) indicated in gray with dashed outline. The variable ϕ can be either pressure p or temperature T.



(a) $x-z$ view of u-CV



(b) $y-z$ view of u-CV, where the "x" indicates the velocity is perpendicular to the plane.

Figure 3.3: The u control volume (u-CV) indicated in gray with dashed outline.

We now define a velocity source term S_u from the terms in the parentheses as

$$S_u^n \equiv \frac{3}{2}\int_\Omega H_u^n \; d\Omega - \frac{1}{2}\int_\Omega H_u^{n-1} \; d\Omega + \int_\Omega F_u \; d\Omega \tag{3.50}$$

Substituting the definition of the source term into Equation (3.49) gives

$$\int_\Omega \widehat{u} \; d\Omega = \int_\Omega u^n \; d\Omega + \frac{\Delta t}{\rho} S_u^n \tag{3.51}$$

The volume integrals in Equation (3.51) can be approximated to second-order accuracy using the midpoint rule [100] as

$$\int_\Omega \widehat{u} \; d\Omega \approx \widehat{u}_{CV} \Delta\Omega_u, \qquad \int_\Omega u^n \; d\Omega \approx u_{CV}^n \Delta\Omega_u \tag{3.52}$$

where subscript CV means at the center of the u-CV and $\Delta\Omega_u$ is the volume of the u-CV. Substituting these approximations into Equation (3.51) and rearranging yields

$$\widehat{u}_{CV} = u_{CV}^n + \Delta t \frac{S_u^n}{\rho \Delta\Omega_u} \tag{3.53}$$

Note that the velocity u_{CV} at the center of the u-CV is the same as the velocity located on the x-face of the grid cell shown in Figure 3.1, and thus $u_{CV} \equiv u_{i,j,k}$. Thus we have

$$\boxed{\widehat{u}_{i,j,k} = u_{i,j,k}^n + \Delta t \frac{S_{u(i,j,k)}^n}{\rho \Delta\Omega_{u(i,j,k)}}} \tag{3.54}$$

Using a similar derivation, the v- and w-momentum components from Equation (3.44) may be integrated to obtain

$$\boxed{\widehat{v}_{i,j,k} = v_{i,j,k}^n + \Delta t \frac{S_{v(i,j,k)}^n}{\rho \Delta\Omega_{v(i,j,k)}}} \tag{3.55}$$

$$\boxed{\widehat{w}_{i,j,k} = w^n_{i,j,k} + \Delta t \frac{S^n_{w(i,j,k)}}{\rho \Delta \Omega_{w(i,j,k)}}} \tag{3.56}$$

Spatial Discretization of Source Term S^n_u

The source term S^n_u from Equation (3.50) will now be evaluated. The body force term F_u is taken as a constant in all simulations and is specified *a priori*; thus the integral is

$$\int_\Omega F_u \, d\Omega = F_u \Delta \Omega_u \tag{3.57}$$

Thus Equation (3.50) is now

$$S^n_u \equiv \frac{3}{2} \int_\Omega H^n_u \, d\Omega - \frac{1}{2} \int_\Omega H^{n-1}_u \, d\Omega + F_u \Delta \Omega_u \tag{3.58}$$

Since the body force F_u is known and the integral of H^{n-1}_u is known from the previous time-step, the only term to be evaluated is the integral of H^n_u, given as follows

$$\int_\Omega H^n_u \, d\Omega = -\int_\Omega \rho \nabla \cdot (u\mathbf{u}) \, d\Omega + \int_\Omega \nabla \cdot (\mu \nabla u) \, d\Omega \tag{3.59}$$

In order to evaluate the volume integrals in Equation (3.59), we now introduce the divergence theorem for a vector field $\mathbf{V}$ defined in a domain Ω with boundary $\partial\Omega$:

$$\int_\Omega \nabla \cdot \mathbf{V} \, d\Omega = \int_{\partial\Omega} \mathbf{V} \cdot \mathbf{n} \, dA \tag{3.60}$$

where $\mathbf{n}$ is a unit normal vector to the differential area dA. The divergence theorem can be used in Equation (3.59) to "convert" the volume integrals to surface integrals. This yields

$$\int_\Omega H^n_u \, d\Omega = \underbrace{-\int_{\partial\Omega} \rho u(\mathbf{u} \cdot \mathbf{n}) \, dA}_{\equiv C^n_u} + \underbrace{\int_{\partial\Omega} \mu \nabla u \cdot \mathbf{n} \, dA}_{\equiv D^n_u} = -C^n_u + D^n_u \tag{3.61}$$

where C_u^n is the convection flux and D_u^n is the diffusion flux for the u-momentum equation. These terms are calculated as shown in Appendix A. The terms C_v^n and D_v^n for the v-momentum equation and C_w^n and D_w^n for the w-momentum equation are calculated similarly.

Spatial Discretization of Momentum Equation for $\mathbf{u}^{n+1}$

Now that the spatial discretization for the first equation of the time-splitting of the momentum equation is complete, we now spatially discretize the second equation of the time-splitting, Equation (3.45). First, we rearrange this equation for $\mathbf{u}^{n+1}$,

$$\mathbf{u}^{n+1} = \widehat{\mathbf{u}} - \frac{\Delta t}{\rho}\nabla p^{n+1} \tag{3.62}$$

or in component form

$$u_{i,j,k}^{n+1} = \widehat{u}_{i,j,k} - \frac{\Delta t}{\rho}\frac{\partial p^{n+1}}{\partial x}\bigg|_u \tag{3.63}$$

$$v_{i,j,k}^{n+1} = \widehat{v}_{i,j,k} - \frac{\Delta t}{\rho}\frac{\partial p^{n+1}}{\partial y}\bigg|_v \tag{3.64}$$

$$w_{i,j,k}^{n+1} = \widehat{w}_{i,j,k} - \frac{\Delta t}{\rho}\frac{\partial p^{n+1}}{\partial z}\bigg|_w \tag{3.65}$$

The pressure gradients are evaluated at the location of the velocity and are discretized using second-order central differences. Thus,

$$\boxed{u_{i,j,k}^{n+1} = \widehat{u}_{i,j,k} - \frac{\Delta t}{\rho}\frac{p_{i+1,j,k}^{n+1} - p_{i,j,k}^{n+1}}{(\Delta x_{i+1} + \Delta x_i)/2}} \tag{3.66}$$

$$\boxed{v_{i,j,k}^{n+1} = \widehat{v}_{i,j,k} - \frac{\Delta t}{\rho}\frac{p_{i,j+1,k}^{n+1} - p_{i,j,k}^{n+1}}{(\Delta y_{j+1} + \Delta y_j)/2}} \tag{3.67}$$

$$\boxed{w_{i,j,k}^{n+1} = \widehat{w}_{i,j,k} - \frac{\Delta t}{\rho}\frac{p_{i,j,k+1}^{n+1} - p_{i,j,k}^{n+1}}{(\Delta z_{k+1} + \Delta z_k)/2}} \tag{3.68}$$

Spatial Discretization of Pressure-Poisson Equation

We begin by writing the Laplacian operator as the divergence of the gradient in the pressure-Poisson equation

$$\nabla^2 p^{n+1} = \nabla \cdot \nabla p^{n+1} = \frac{\rho}{\Delta t} (\nabla \cdot \widehat{\mathbf{u}}) \tag{3.69}$$

We apply the Finite Volume Method by integrating over a domain Ω. Since we are solving for the pressure, the domain Ω is the scalar control volume of Figure 3.2. The integration is written as

$$\int_{\Omega} \nabla \cdot \nabla p^{n+1} \, d\Omega = \frac{\rho}{\Delta t} \int_{\Omega} \nabla \cdot \widehat{\mathbf{u}} \, d\Omega \tag{3.70}$$

and then applying the divergence theorem yields

$$\int_{\partial\Omega} \nabla p^{n+1} \cdot \mathbf{n} \, dA = \frac{\rho}{\Delta t} \int_{\partial\Omega} \widehat{\mathbf{u}} \cdot \mathbf{n} \, dA \tag{3.71}$$

The boundary $\partial\Omega$ is the union of the six faces of the scalar control volume in Figure (3.2). Thus Equation (3.71) can be written as

$$\sum_{faces} \int_{A_{face}} \nabla p^{n+1} \cdot \mathbf{n} \, dA = \frac{\rho}{\Delta t} \sum_{faces} \int_{A_{face}} \widehat{\mathbf{u}} \cdot \mathbf{n} \, dA \tag{3.72}$$

If we note that the mass flow rate based on the provisional velocity field is

$$\widehat{\dot{m}}_{face} = \int_{A_{face}} \rho \widehat{\mathbf{u}} \cdot \mathbf{n} \, dA \tag{3.73}$$

then we can write Equation (3.72) as

$$\sum_{faces} \int_{A_{face}} \nabla p^{n+1} \cdot \mathbf{n} \, dA = \frac{1}{\Delta t} \sum_{faces} \widehat{\dot{m}}_{face} \tag{3.74}$$

Let us now evaluate each of the integrals on the left-hand-side for each of the six faces and discretize the pressure gradient using central differencing:

$$\int\limits_{A_{x_+}} \nabla p^{n+1} \cdot \mathbf{i}\, dA = \int\limits_{A_{x_+}} \left.\frac{\partial p^{n+1}}{\partial x}\right|_{x_+\ face} dA = \frac{p^{n+1}_{i+1,j,k} - p^{n+1}_{i,j,k}}{(\Delta x_i + \Delta x_{i+1})/2} A_{x_+} \tag{3.75}$$

$$\int\limits_{A_{x_-}} \nabla p^{n+1} \cdot -\mathbf{i}\, dA = \int\limits_{A_{x_-}} \left.-\frac{\partial p^{n+1}}{\partial x}\right|_{x_-\ \text{face}} dA = \frac{p^{n+1}_{i,j,k} - p^{n+1}_{i-1,j,k}}{(\Delta x_i + \Delta x_{i-1})/2} A_{x_-} \tag{3.76}$$

$$\int\limits_{A_{y_+}} \nabla p^{n+1} \cdot \mathbf{j}\, dA = \int\limits_{A_{y_+}} \left.\frac{\partial p^{n+1}}{\partial y}\right|_{y_+\ \text{face}} dA = \frac{p^{n+1}_{i,j+1,k} - p^{n+1}_{i,j,k}}{(\Delta y_j + \Delta y_{j+1})/2} A_{y_+} \tag{3.77}$$

$$\int\limits_{A_{y_-}} \nabla p^{n+1} \cdot -\mathbf{j}\, dA = \int\limits_{A_{y_-}} \left.-\frac{\partial p^{n+1}}{\partial y}\right|_{y_-\ \text{face}} dA = \frac{p^{n+1}_{i,j,k} - p^{n+1}_{i,j-1,k}}{(\Delta y_j + \Delta y_{j-1})/2} A_{y_-} \tag{3.78}$$

$$\int\limits_{A_{z_+}} \nabla p^{n+1} \cdot \mathbf{k}\, dA = \int\limits_{A_{z_+}} \left.\frac{\partial p^{n+1}}{\partial z}\right|_{z_+\ \text{face}} dA = \frac{p^{n+1}_{i,j,k+1} - p^{n+1}_{i,j,k}}{(\Delta z_k + \Delta z_{k+1})/2} A_{z_+} \tag{3.79}$$

$$\int\limits_{A_{z_-}} \nabla p^{n+1} \cdot -\mathbf{k}\, dA = \int\limits_{A_{z_-}} \left.-\frac{\partial p^{n+1}}{\partial z}\right|_{z_-\ \text{face}} dA = \frac{p^{n+1}_{i,j,k} - p^{n+1}_{i,j,k-1}}{(\Delta z_k + \Delta z_{k-1})/2} A_{z_-} \tag{3.80}$$

Substituting Equations (3.75) to (3.80) into Equation (3.74) and simplifying we obtain the final form of the discrete pressure-Poisson equation:

$$\boxed{a_e p^{n+1}_{i+1,j,k} + a_w p^{n+1}_{i-1,j,k} + a_n p^{n+1}_{i,j+1,k} + a_s p^{n+1}_{i,j-1,k} + a_h p^{n+1}_{i,j,k+1} + a_l p^{n+1}_{i,j,k-1} - a_p p^{n+1}_{i,j,k} = \frac{1}{\Delta t} \sum_{faces} \widehat{\dot{m}}_{face}} \tag{3.81}$$

where the coefficients use compass notation for the subscripts (east, west, north, south, high, low). These coefficients are

$$a_e = \frac{A_{x_+}}{(\Delta x_i + \Delta x_{i+1})/2} = \frac{\Delta y_j \Delta z_k}{(\Delta x_i + \Delta x_{i+1})/2} \tag{3.82}$$

$$a_w = \frac{A_{x_-}}{(\Delta x_i + \Delta x_{i-1})/2} = \frac{\Delta y_j \Delta z_k}{(\Delta x_i + \Delta x_{i-1})/2} \tag{3.83}$$

$$a_n = \frac{A_{y_+}}{(\Delta y_j + \Delta y_{j+1})/2} = \frac{\Delta x_i \Delta z_k}{(\Delta y_j + \Delta y_{j+1})/2} \tag{3.84}$$

$$a_s = \frac{A_{y_-}}{(\Delta y_j + \Delta y_{j-1})/2} = \frac{\Delta x_i \Delta z_k}{(\Delta y_j + \Delta y_{j-1})/2} \tag{3.85}$$

$$a_h = \frac{A_{z_+}}{(\Delta z_k + \Delta z_{k+1})/2} = \frac{\Delta x_i \Delta y_j}{(\Delta z_k + \Delta z_{k+1})/2} \tag{3.86}$$

$$a_l = \frac{A_{z_-}}{(\Delta z_k + \Delta z_{k-1})/2} = \frac{\Delta x_i \Delta y_j}{(\Delta z_k + \Delta z_{k-1})/2} \tag{3.87}$$

$$a_p = a_e + a_w + a_n + a_s + a_h + a_l \tag{3.88}$$

Equation (3.81) is the discrete equation that is used to update the pressure in a given cell (i, j, k) to the next time-level $n + 1$. This is an implicit equation and thus represents a system of linear equations that must be solved simultaneously. The strategy for solving it is discussed in the next section.

3.3.4 Numerical Solution of Pressure-Poisson Equation

Iterative Linear Solver

The discrete pressure-Poisson equation represents a system of linear equations. A number of iterative linear solvers are available, but the selected solver should suit the parallel nature of the GPU. Probably the most obvious choice is Jacobi iteration since it is naturally parallel, but it has a poor convergence rate and was not used. A better choice is red-black Successive Over-Relaxation (SOR), which uses a coloring strategy to make the numerical solution parallel and has a better convergence rate than Jacobi iteration. The name "red-black" is used because the mesh is colored like a checkerboard, as shown in Figure 3.4. The pressure stencil indicates that the pressures in red cells are a function only of the pressures in the black cells, and vice-versa. Thus, by solving for one color at a time, each pressure update in a given cell is independent, and the color update can be multithreaded.

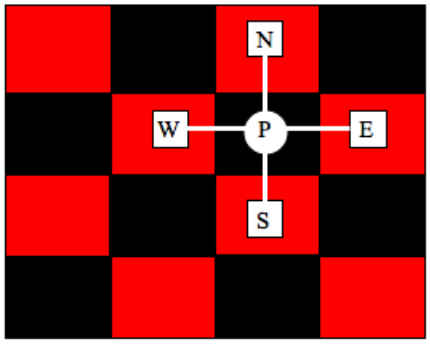


Figure 3.4: Mesh with red-black coloring for pressure.

For a given color (red or black), the iterative update for red-black SOR is a two-step process. First, we find a provisional value of the pressure $\widehat{p}_{i,j,k}$ using

$$\widehat{p}_{i,j,k} = \frac{a_e p^s_{i+1,j,k} + a_w p^s_{i-1,j,k} + a_n p^s_{i,j+1,k} + a_s p^s_{i,j-1,k} + a_h p^s_{i,j,k+1} + a_l p^s_{i,j,k-1} - \frac{1}{\Delta t}\sum_{faces} \widehat{m}_{face}}{a_p} \tag{3.89}$$

where the superscript s is the sweep or iteration number. More concisely, we can state this as

$$\widehat{p}_{i,j,k} = \frac{1}{a_p}\left(\sum_{nb} a_{nb} p^s_{nb} - \frac{1}{\Delta t}\sum_{faces} \widehat{m}_{face}\right) \tag{3.90}$$

where the subscript nb refers to the neighbors of the cell at (i, j, k). Then, we over-relax using

$$p^{s+1}_{i,j,k} = \omega \widehat{p}_{i,j,k} + (1-\omega) p^s_{i,j,k} \tag{3.91}$$

where ω is the over-relaxation factor ($1 < \omega < 2$). Equation (3.89) looks like Jacobi iteration, but the difference lies in the fact that it is applied to only one color at a time. First, we apply SOR to the red pressures to advance them to iteration $s+1$, which are a function of only the black neighbors at iteration s. Then we apply SOR to the black pressures to advance them to iteration $s+1$, which are a function of only the red neighbors that are now at iteration $s+1$. The algorithm for red-black SOR is given in Algorithm 2.

Algorithm 2: Red-Black Successive Over-Relaxation

for $s = 1$ ***to*** *total number of sweeps* **do**
  forall *cells in domain* **do**
    if (i, j, k) *is a red cell* **then**
      $\widehat{p}_{i,j,k} = \frac{1}{a_p}\left(\sum_{nb \in black} a_{nb} p^s_{nb} - \frac{1}{\Delta t}\sum_{faces} \widehat{\dot{m}}_{face}\right)$
      $p^{s+1}_{i,j,k} = \omega \widehat{p}_{i,j,k} + (1-\omega) p^s_{i,j,k}$
    end
  end
  forall *cells in domain* **do**
    if (i, j, k) *is a black cell* **then**
      $\widehat{p}_{i,j,k} = \frac{1}{a_p}\left(\sum_{nb \in red} a_{nb} p^{s+1}_{nb} - \frac{1}{\Delta t}\sum_{faces} \widehat{\dot{m}}_{face}\right)$
      $p^{s+1}_{i,j,k} = \omega \widehat{p}_{i,j,k} + (1-\omega) p^s_{i,j,k}$
    end
  end
end

Multigrid Algorithm

The numerical solution of the pressure-Poisson equation is the most computationally expensive stage of the simulation (accounting for approximately 2/3 of the total simulation time); this motivates accelerating the convergence of the PPE solution. A geometric multigrid method is employed for this purpose using a V-cycle, as shown in Figure 3.5. Starting on the finest mesh, red-black SOR is applied to smooth the error in the pressure, then the pressure residual is calculated and restricted to the next coarser mesh level. This is repeated until the coarsest level is reached. Then the pressure solution is prolongated to the next finer mesh level and red-black SOR is again applied to further smooth the error in pressure. This is repeated until the finest mesh is reached, which completes one V-cycle. The multigrid algorithm is shown in Algorithm 3. For each time-level, multiple V-cycles are performed to further reduce the pressure residual. The exact number of V-cycles per time-step varies from problem to problem, but typically 3 to 5 V-cycles were used, which was a good compromise

between convergence rate and computational time.

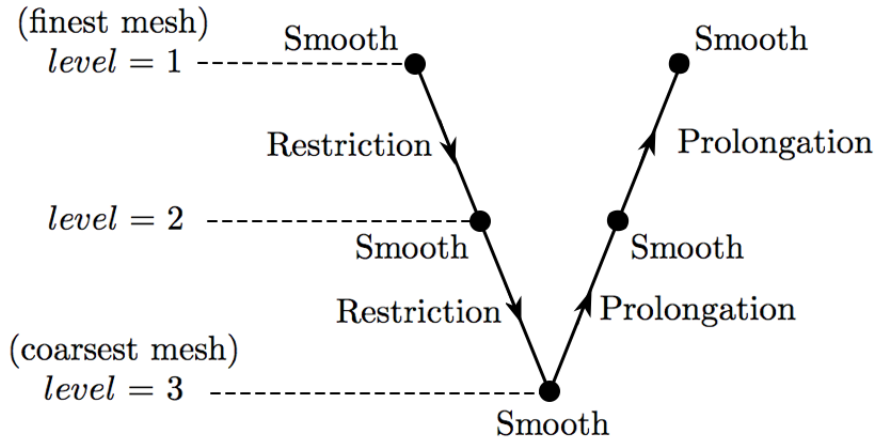


Figure 3.5: Multigrid V-cycle. Only three mesh levels ($ngrid = 3$) are shown for illustrative purposes.

Algorithm 3: Multigrid Algorithm for Solving the PPE

```
for c = 1 to total number of v-cycles do
    for level = 1 to ngrid do
        smooth pressure using red-black SOR (Algorithm 2)
        if level ≠ ngrid then
            calculate residual of pressure
            restrict pressure to coarser grid level
        end
    end
    if ngrid ≠ 1 then
        for level = ngrid to 2 do
            prolongate pressure to finer grid level
            if level ≠ 2 then
                smooth pressure using red-black SOR (Algorithm 2)
            end
        end
    end
end
```

Mesh resolutions vary from one level to another by powers of two in each direction: a finer grid has twice as many cells in each coordinate direction as the next coarser grid in the V-cycle. Starting from the finest grid, geometric agglomeration was used to calculate the grid spacings Δx, Δy, and Δz on the coarser mesh levels. For example, Δx of a given coarse cell is the sum of the two Δx mesh spacings corresponding to the two finer cells at

that location. Thus eight cells in a finer mesh correspond to one cell in a coarser mesh. Agglomeration was used to ensure that the coarser cell boundaries precisely match the finer cell boundaries when grid stretching is used.

3.3.5 Discretization of the Energy Equation

We start by writing the energy equation, Equation (3.3), with the time derivative on the left-hand-side

$$\rho c_p \frac{\partial T}{\partial t} = -\rho c_p \nabla \cdot (\mathbf{u}T) + \nabla \cdot (k \nabla T) \tag{3.92}$$

and combine the convection and diffusion terms into a single term H_T as

$$H_T \equiv -\rho c_p \nabla \cdot (\mathbf{u}T) + \nabla \cdot (k \nabla T) \tag{3.93}$$

Thus, we have

$$\rho c_p \frac{\partial T}{\partial t} = H_T \tag{3.94}$$

Writing this equation discretely in time using the explicit second-order accurate Adams-Bashforth method yields

$$\rho c_p \frac{T^{n+1} - T^n}{\Delta t} = \frac{3}{2} H_T^n - \frac{1}{2} H_T^{n-1} \tag{3.95}$$

and solving for the updated temperature

$$T^{n+1} = T^n + \frac{\Delta t}{\rho c_p} \left(\frac{3}{2} H_T^n - \frac{1}{2} H_T^{n-1} \right) \tag{3.96}$$

Now that the time discretization is complete, we discretize in space using the Finite Volume Method by integrating Equation (3.96) over the scalar-control volume (see Figure 3.2),

defined as a region Ω

$$\int_{\Omega} T^{n+1}\ d\Omega = \int_{\Omega} T^{n}\ d\Omega + \frac{\Delta t}{\rho c_p}\left(\frac{3}{2}\int_{\Omega} H_T^{n}\ d\Omega - \frac{1}{2}\int_{\Omega} H_T^{n-1}\ d\Omega\right) \tag{3.97}$$

We now define a velocity source term S_T from the terms in the parentheses as

$$S_T^n \equiv \frac{3}{2}\int_{\Omega} H_T^{n}\ d\Omega - \frac{1}{2}\int_{\Omega} H_T^{n-1}\ d\Omega \tag{3.98}$$

Substituting the definition of the source term into Equation (3.97) gives

$$\int_{\Omega} T^{n+1}\ d\Omega = \int_{\Omega} T^{n}\ d\Omega + \frac{\Delta t}{\rho c_p} S_T^n \tag{3.99}$$

The volume integrals can be approximated to second-order accuracy using the midpoint rule as

$$\int_{\Omega} T^{n+1}\ d\Omega \approx T_{CV}^{n+1}\Delta\Omega_T, \qquad \int_{\Omega} T^{n}\ d\Omega \approx T_{CV}^{n}\Delta\Omega_T \tag{3.100}$$

where subscript CV means at the center of the scalar-CV and $\Delta\Omega_T$ is the volume of the scalar-CV. Substituting these approximations into Equation (3.99) and rearranging yields

$$T_{CV}^{n+1} = T_{CV}^{n} + \Delta t \frac{S_T^n}{\rho c_p \Delta\Omega_T} \tag{3.101}$$

Note that the temperature T_{CV} at the center of the scalar-CV is the same as $T_{i,j,k}$. Thus we have

$$\boxed{T_{i,j,k}^{n+1} = T_{i,j,k}^{n} + \Delta t \frac{S_T^n}{\rho c_p \Delta\Omega_T}} \tag{3.102}$$

In order to know the source term S_T^n from Equation (3.98), we need to compute

$$\int_\Omega H_T^n \, d\Omega = -\int_\Omega \rho c_p \nabla \cdot (\mathbf{u}T) \, d\Omega + \int_\Omega \nabla \cdot (k \nabla T) \, d\Omega \tag{3.103}$$

which we do by using the divergence theorem to get

$$\int_\Omega H_T^n \, d\Omega = \underbrace{-\int_{\partial\Omega} \rho c_p T(\mathbf{u} \cdot \mathbf{n}) \, dA}_{\equiv C_T^n} + \underbrace{\int_{\partial\Omega} k \nabla T \cdot \mathbf{n} \, dA}_{\equiv D_T^n} = -C_T^n + D_T^n \tag{3.104}$$

where boundary $\partial\Omega$ is the boundary of the scalar-CV, $\mathbf{n}$ is a unit normal vector to the scalar-CV surface, C_T^n is the convection flux, and D_T^n is the diffusion flux for the energy equation. Since the convection and diffusion fluxes of the energy equation are calculated in a fashion similar to the fluxes of the u-momentum equation that was shown in Appendix A, these details are omitted.

3.4 Implementation of Flow Solver on a Single GPU

3.4.1 Multithreading

The Navier-Stokes solver was written using CUDA (Compute Unified Device Architecture). CUDA is a programming paradigm developed by NVIDIA, and is essentially the C programming language with extensions added for accessing the GPU. In the algorithm, the grid generation data, initial conditions, and boundary conditions were created on the CPU and then these data were copied from the CPU to GPU. The time-stepping loop was executed on the CPU, and for each routine a separate kernel was launched that performed the computations on the GPU. After the final time-level is reached, the time-stepping loop is exited and the final flow-field on the GPU is copied back to the CPU and written to a file.

To understand how the GPU threads map to the computational mesh, consider Figure

3.6, which shows a mesh of the internal cells with dimensions of $nx[level]$ x $ny[level]$ x $nz[level]$. By internal cells we mean the cells inside the boundaries where flow variables are updated. The arrays $nx[level]$, $ny[level]$, and $nz[level]$ contain the number of mesh cells in each direction for the given mesh level in the multigrid V-cycle. The indices of the internal cells range from $(i, j, k) = (2, 2, 2)$ to $(i, j, k) = (nx[level]+1, ny[level]+1, nz[level]+1)$. The boundary cells (which are not shown in Figure 3.6) lie along the planes $i = 1, j = 1, k = 1$ and planes $i = nx[level] + 2, j = ny[level] + 2, k = nz[level] + 2$.

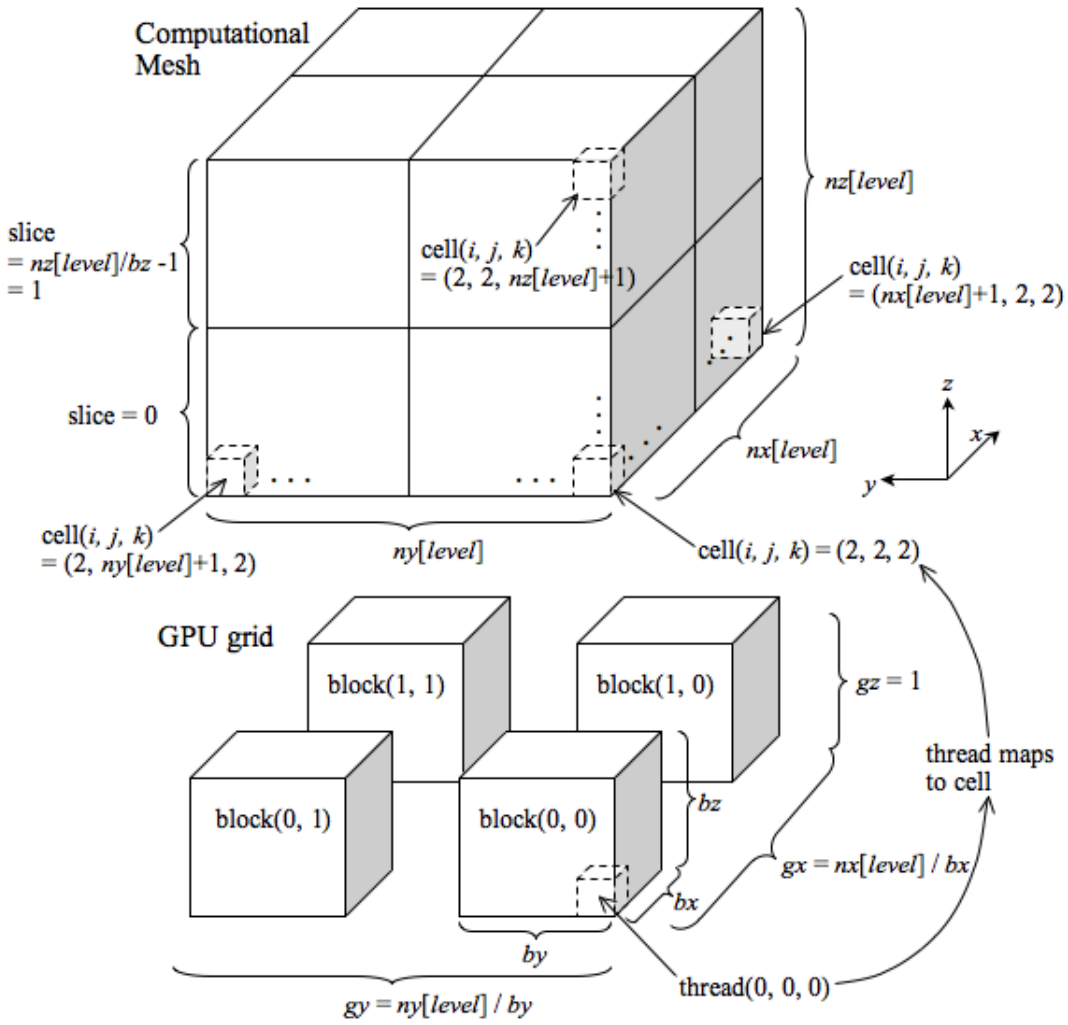


Figure 3.6: Correspondence between GPU grid and computational mesh.

The GPU grid dimensions are (gx, gy, gz) and each block has dimensions (bx, by, bz).

The GPU grid dimensions gx and gy were calculated by dividing the dimensions of the computational mesh on the current mesh level by the block size. Thus, while performing multigrid, the GPU grid dimensions are changed to accommodate the size of the current computational mesh level. This idea is implemented in the example code shown in Table 3.1, where the execution configuration in the main program (on the CPU) is changed as a function of the mesh level when calling a kernel for the GPU.

Since the thread indices always start at zero, they are incremented by two so that they map to the computational mesh. The GPU grid and computational mesh have the same dimensions in the x- and y-directions, so that the threads map one-to-one with the cells. However, due to the fact that the GPU grid can only have a z-dimension equal to one requires the threads to be reused for other cells in that direction. This is done by operating on slices of the computational mesh, where a thread for an (i, j) location updates one cell in each slice. Thus a single thread operates on multiple cells, moving in the k direction in a column for fixed (i, j). No threads are assigned to the boundary cells, since no updating is performed there.

The mapping concept shown in Figure 3.6 is implemented in the kernel code shown in Table 3.1. The thread indices (`tx`, `ty`, `tz`) are computed from the built-in GPU variables `threadIdx`, `blockIdx`, `blockDim`, which are the thread index in a given block, block index of a given block, and block dimension of a given block, respectively. In order to update all the cells in the mesh, a loop that goes over all slices was used inside the kernel to allow a thread for an (i, j) location to update one cell in each slice. As the slice index varies in the loop, so does the k index for the cells that the thread operates on.

3.4.2 Optimization

As mentioned previously, the numerical solution of the pressure-Poisson equation consumes the most computing time, thus it should be the primary focus for performance optimization. In an effort to decrease memory access times, textures were used to fetch the pressure

Table 3.1: Example code showing how multigrid is handled in the execution configuration and how threads are mapped to computational cells.

```
  int main(void)
{
    ...
    for( n = 1; n<=ngrid; n++)
  {
   dim3 block(bx,by,bz);
   dim3 grid(nx[n]/bx,ny[n]/by);
   kernel<<<grid, block>>>( ... );
  }
  ...
} // end main

  __global__ void kernel( ... )
{
   // global thread indices
   tx = threadIdx.x + blockIdx.x * blockDim.x;
   ty = threadIdx.y + blockIdx.y * blockDim.y;
   tz = threadIdx.z;

   // convert thread indices to mesh indices
   i = tx + 2;
   j = ty + 2;

   for (slice=0; slice<=nz[n]/blockDim.z-1; slice++) {
     k = tz + slice * blockDim.z + 2;
     ...
     computations
     ...
   } // end slice
}  // end kernel
```

data from global memory, which decreased overall code execution time by approximately 10 percent. Textures enhance performance by avoiding uncoalesced loads from global memory [7]. Performance is very sensitive to block size, so this is another area of code optimization. Block sizes must evenly divide into the mesh dimensions for each mesh level for multigrid (as shown in Figure 3.6). A block size that can accommodate the coarsest level could be selected, which would accommodate all finer mesh levels. However, this may not yield optimal GPU performance since the block size is small (smaller than the warp size). A compromise between accommodating each mesh level and performance was found by using two block sizes: a block size for the finer meshes and a block size for the coarser meshes. Most of the computation occurs on the finer meshes (first one or two mesh levels in the V-cycle), and thus the block sizes for these levels were tuned for optimal performance. It was found that for most problems a good block size is $(bx, by, bz) = (32, 1, 8)$. This can change from problem to problem, so it is best to experiment to determine the optimal sizes. For the coarser meshes, a smaller block size was used so that the mesh resolution would be evenly divisible by the block size. The block size must be small enough to accommodate the coarsest mesh, which would typically be $nx[ngrid]$ x $ny[ngrid]$ x $nz[ngrid]$ = 4 x 4 x 4, thus the coarse block size was $(bx, by, bz) = (4, 4, 4)$. The smaller block size delivers poor performance, but this only occurs on the coarse levels which do not have appreciable computing times, so the effect is small.

3.5 Implementation of Flow Solver on a Multiple GPUs

In order to improve the speed-up and scale to larger problem sizes, a multi-GPU implementation of the Navier-Stokes solver was created. In order to use multiple GPUs, it is good practice to use multiple CPUs to manage the GPU kernel calls, where each CPU is assigned a GPU. For this purpose, this implementation used POSIX Threads [2], which is a program-

ming paradigm for using multiple CPU threads that can be used for parallel computing. The concept is illustrated in Figure 3.7, where a master thread creates worker threads (one for each CPU) and these threads execute instructions in parallel. Once all threads finish, they join and the process completes. This paradigm is coupled with CUDA such that a POSIX thread is launched for each CPU core and then "assigned" to a GPU.

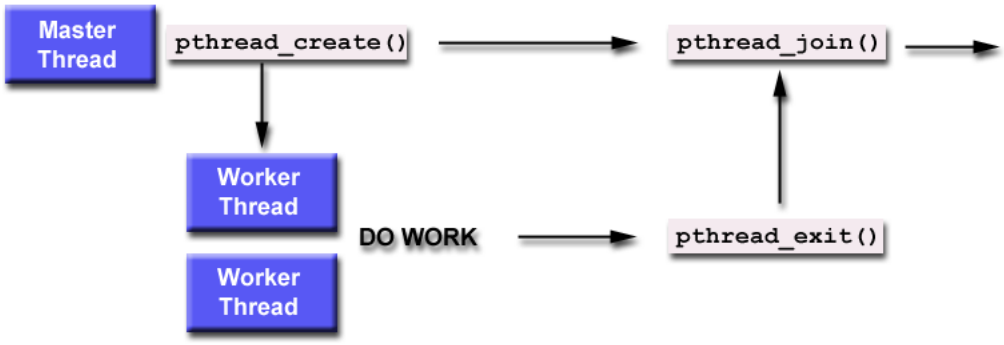


Figure 3.7: POSIX thread model [101].

POSIX Threads use the shared memory model of parallel computing, where all CPUs have access to the same RAM. This technique is ideal if one is using a single workstation, which is the present case. In particular, the present work was conducted using a 64-bit Linux workstation with 16 GB RAM, an AMD 2.6 GHz quad-core processor, and four NVIDIA Tesla C1060 GPUs; an interior picture of the multi-GPU arrangement in the workstation is shown in Figure 3.8. With four Tesla C1060 GPUs, the machine is capable of nearly 4 TeraFLOPs in single precision and 312 GFLOPS in double precision. In addition, the total GPU RAM is 16 GB, the same as for the CPU.

In order to divide the computational work among the GPUs, a domain decomposition is performed such that the number of subdomains equals the number of GPUs available. Each worker thread is responsible for its own subdomain and executes the time-stepping loop in parallel with other threads. While in the time-stepping loop, each worker thread will launch kernels that run on its assigned GPU. Boundary data must be communicated between the subdomains each time the data in the interior are updated. In CUDA, there is no way to

Figure 3.8: Multi-GPU configuration inside workstation

communicate data directly between GPUs, so the data must first be copied back to the CPU and then distributed. This was implemented using a communication routine that was called after every variable update. The communication routine performs the following in order:

1. Pack boundary data into a data structure on the GPU
2. Copy boundary data from GPU to its assigned CPU, where it is loaded into a local array for that CPU
3. Send boundary data from worker thread to master thread, where it is loaded into a global array
4. Synchronize worker threads to ensure all data have been sent to master thread
5. Each worker thread receives boundary data from master thread
6. Copy boundary data from CPU to GPU
7. Unpack boundary data on GPU
8. Synchronize worker threads

In order to hide the communication time, it is a good practice to try to perform computations while communicating boundary data. In CUDA, one can create "streams" for doing this, where one stream performs computations and the other performs communication simultaneously. This was implemented but did not show much improvement in the compute time, presumably due to lack of optimization or incorrect implementation. In the future it is recommended to investigate this further to optimize the code. In addition, it should be noted that the multi-GPU solver was only used for experimental purposes and never reached the production-ready stage due to time limitations in the research. In the future, it is highly recommended that multi-GPU implementations be explored due to the vast scaling potential. The performance of the present multi-GPU solver will be shown in the next chapter for a benchmark problem.

3.6 Ghost Cell Method for Flow Past Complex Geometries

3.6.1 Overview

A 3D ghost cell method is developed to handle flow past complex geometries. The boundary of the complex geometry is represented by triangular segments, and these segments are immersed within a staggered, Cartesian mesh. Since a staggered mesh is used, each cell has three velocity points at the cell faces (for u, v, w) and one scalar point at the cell center (for p, T). The velocity and scalar points that are outside the immersed boundary are considered in the fluid and are updated according to the Navier-Stokes equations; these will be called *fluid points*. The velocity and scalar points that are inside the immersed boundary are considered in the solid. The points in the solid are classified in two ways: if a point in the solid has any neighbor point that is in the fluid, then it is a *ghost point* and if a point in the solid does not have any neighboring fluid points then it is an *interior point*. Thus, the

ghost points are adjacent to the immersed boundary, and the interior points are away from the immersed boundary.

Figure 3.9 shows a 2D view of a Cartesian mesh with an arbitrary immersed boundary represented by segments. The u-velocity points are shown as an example. The fluid points are tagged with a 1, the ghost points are tagged with a 2, and the interior points are tagged with a 0. This tagging is used in the software implementation of the method to classify the points. Similar tagging is used for all other variables. Since variables have different locations on a cell, a given cell could have some points as ghost points and others as fluid points, depending on how the immersed boundary intersects the cell. To account for this, separate tagging is used for the variables u, v, w, T.

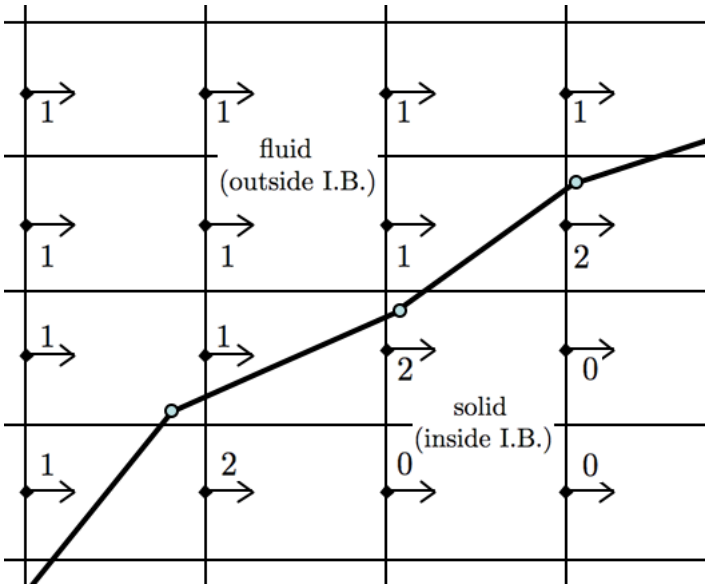


Figure 3.9: 2D view of mesh showing tags for u-velocity points.

The velocity and temperature at the ghost points are set at each time-step such that desired boundary conditions are satisfied at the immersed boundary. This is achieved by mirroring each ghost point at location $\vec{x}_G$ across the closest surface segment along the segment's unit normal vector $\widehat{n}$, which places a point in the fluid called a *mirror point* at location $\vec{x}_M$ (see Figure 3.10). The velocities or temperatures at nearest neighbors to the mirror point are interpolated onto the mirror point.

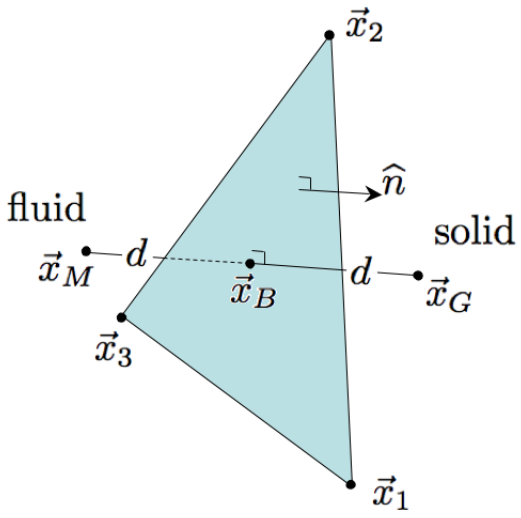


Figure 3.10: Triangular segment with surface unit normal $\widehat{n}$.

Then boundary conditions are applied using the mirror point; for example, for the velocity we apply the no-slip condition at the immersed boundary:

$$\frac{\phi_G + \phi_M}{2} = \phi_B \Rightarrow \phi_G = 2\phi_B - \phi_M \tag{3.105}$$

where ϕ can be u, v, or w. For the temperature we apply the adiabatic condition along the normal vector (in the direction n):

$$\frac{\partial T}{\partial n} = \frac{T_G - T_M}{2d} = 0 \Rightarrow T_G = T_M \tag{3.106}$$

where d is the perpendicular distance from the ghost point to the surface segment, and $2d$ is the perpendicular distance from the ghost point to the mirror point.

The ghost points for pressure are updated using the pressure-Poisson equation, just like the fluid points. As a consequence, local mass conservation is preserved in the ghost cells and there is no mass flux across the immersed boundary. An alternative method for updating the ghost point pressures is interpolating pressures from the fluid points to a mirror point and applying a Neumann boundary condition across the surface segments, as used in [73, 102].

This method, however, can produce mass fluxes across the immersed boundary [76] and thus was not used.

The present method is similar to the mirroring immersed boundary method (MIB) of Mark and van Wachem [74]. One main difference is that the present work introduces a way to avoid a feed-back instability that can occur if a ghost point value is set using an interpolation that includes that same ghost point, but using the value set from the previous time-level. This can happen if the immersed boundary is close to the ghost point, making it one of the nearest-neighbors selected for the interpolation of the mirror point. This concept was previously introduced in the 2D ghost cell implementation of Shinn et al. [76].

The present ghost cell method has been applied to the micro-ramp geometry discussed earlier. Figure 3.11 shows the Cartesian mesh of micro-ramp (in blue) where the cells lie within the boundaries of the exact micro-ramp shape (outlined in black). The black lines represent the edges of the triangular segments for the ghost cell method, where only three segments are needed (left side, right side, and top). Figure 3.12 illustrates a zoomed view of Cartesian mesh of the micro-ramp (on leeward side at the midspan) where the top segment is passing through the cells. The cells are colored according to the classification of the cell-centered point, where the green cells are fluid cells (flag=1), the blue cells are interior cells (not solved, flag=0), and the red cells are the ghost cells (flag=2) updated using the immersed boundary method.

3.6.2 Trilinear Interpolation

Consider a mirror point ϕ_M at location (x_M, y_M, z_M), where ϕ can be u, v, w, or T. Eight nearest-neighbor points are located, which creates an interpolation box around the mirror point, as shown in Figure 3.13. If $\phi = u$, then the corners of the interpolation box are the nearest x-face centers. Likewise, if $\phi = v$, then the corners of the interpolation box are the nearest y-face centers and if $\phi = w$, then the corners of the interpolation box are the nearest z-face centers. In addition, if $\phi = T$, then the corners of the interpolation box are

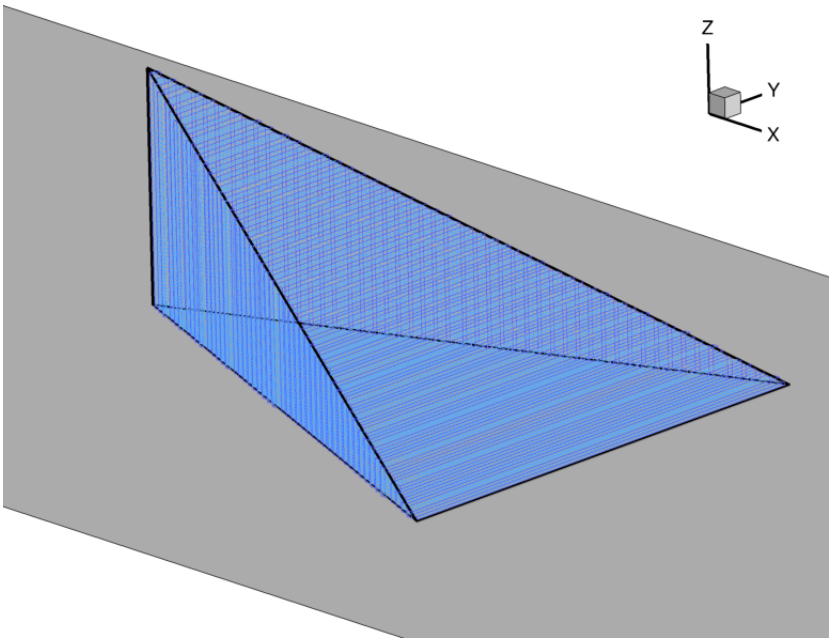


Figure 3.11: Cartesian mesh of micro-ramp (blue) with exact micro-ramp shape outlined in black. The black lines represent the edges of the triangular segments for the ghost cell method.

the nearest cell centers.

The trilinear interpolation for the mirror point $\phi_M(x_M, y_M, z_M)$ within the interpolation box formed by eight nearest-neighbor points $\phi_1, ..., \phi_8$ is

$$\begin{aligned}\phi_M = &\underbrace{\alpha\beta\gamma}_{f_{\phi,1}}\phi_1 + \underbrace{(1-\alpha)\beta\gamma}_{f_{\phi,2}}\phi_2 + \underbrace{\alpha(1-\beta)\gamma}_{f_{\phi,3}}\phi_3 + \underbrace{(1-\alpha)(1-\beta)\gamma}_{f_{\phi,4}}\phi_4 + \underbrace{\alpha\beta(1-\gamma)}_{f_{\phi,5}}\phi_5 \\ &+ \underbrace{(1-\alpha)\beta(1-\gamma)}_{f_{\phi,6}}\phi_6 + \underbrace{\alpha(1-\beta)(1-\gamma)}_{f_{\phi,7}}\phi_7 + \underbrace{(1-\alpha)(1-\beta)(1-\gamma)}_{f_{\phi,8}}\phi_8 \end{aligned} \tag{3.107}$$

where

$$\alpha = \frac{x_+ - x_M}{x_+ - x_-} \qquad \beta = \frac{y_+ - y_M}{y_+ - y_-} \qquad \gamma = \frac{z_+ - z_M}{z_+ - z_-} \tag{3.108}$$

$$x_- \leq x_M \leq x_+ \qquad y_- \leq y_M \leq y_+ \qquad z_- \leq z_M \leq z_+ \tag{3.109}$$

We can write Equation (3.107) concisely as

$$\phi_M = f_{\phi,1}\phi_1 + f_{\phi,2}\phi_2 + f_{\phi,3}\phi_3 + f_{\phi,4}\phi_4 + f_{\phi,5}\phi_5 + f_{\phi,6}\phi_6 + f_{\phi,7}\phi_7 + f_{\phi,8}\phi_8 \tag{3.110}$$

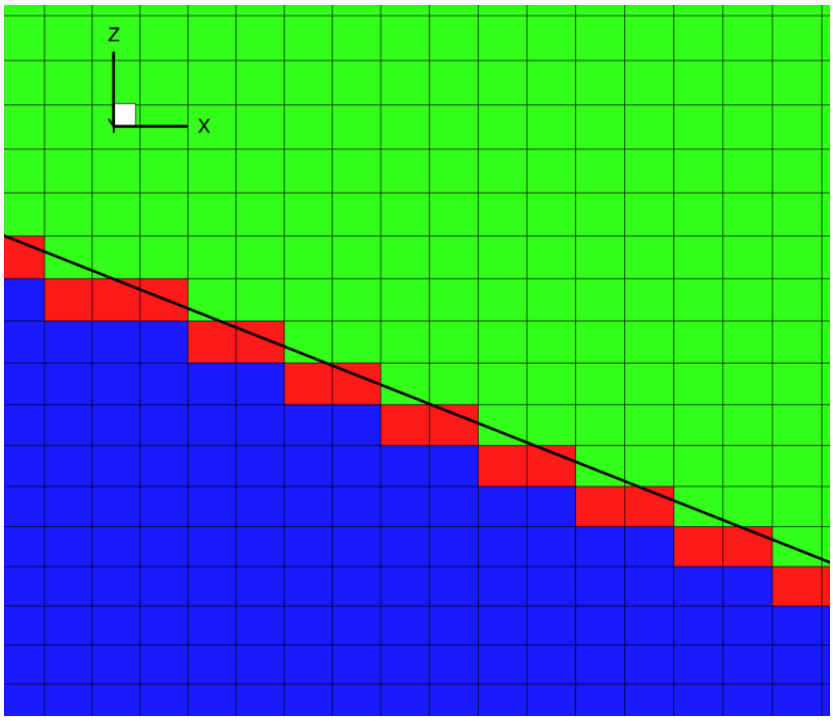


Figure 3.12: Zoomed view of Cartesian mesh of micro-ramp (on leeward side) with immersed boundary passing through the cells. The cells are colored according to the classification of the cell-centered point, where the green cells are fluid cells (flag=1), the blue cells are interior cells (not solved, flag=0), and the red cells are the ghost cells (flag=2) updated using the immersed boundary method.

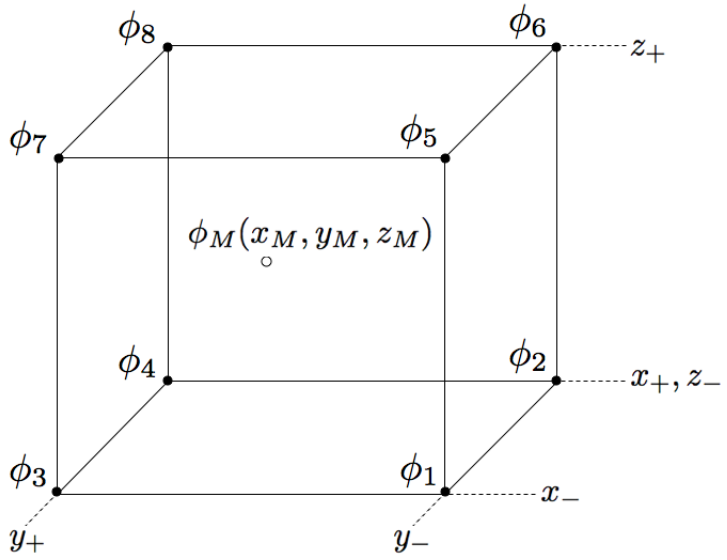


Figure 3.13: Interpolation box using eight points to interpolate for the mirror point.

or

$$\boxed{\phi_M = \sum_{i=1}^{8} f_{\phi,i}\phi_i} \tag{3.111}$$

If one of the eight interpolation points (say the jth) is the ghost point being mirrored across the segment (primary ghost point), then it should be eliminated from Equation (3.111) to prevent a feed-back instability caused by directly using the ghost value in the interpolation. This is performed differently for the velocity and temperature interpolations due to different boundary conditions; each will be considered separately below.

Removing Primary Ghost Point from Velocity Interpolation

The following procedure will remove the primary ghost point value from the velocity interpolation. First removing it from the summation yields

$$\phi_M = \sum_{i=1}^{8} f_{\phi,i}\phi_i = f_{\phi,j}\phi_j + \sum_{i=1,i\neq j}^{8} f_{\phi,i}\phi_i \tag{3.112}$$

Then, from the no-slip condition of Equation 3.105 we have $\phi_j = \phi_G = 2\phi_B - \phi_M$, and substituting into Equation (3.112) yields

$$\phi_M = f_{\phi,j}(2\phi_B - \phi_M) + \sum_{i=1,i\neq j}^{8} f_{\phi,i}\phi_i \tag{3.113}$$

This is an implicit equation for ϕ_M, and can be solved for ϕ_M as

$$\boxed{\phi_M = \frac{2f_{\phi,j}}{1+f_{\phi,j}}\phi_B + \sum_{i=1,i\neq j}^{8} \frac{f_{\phi,i}}{1+f_{\phi,j}}\phi_i} \tag{3.114}$$

Removing Primary Ghost Point from Temperature Interpolation

The following procedure will remove the primary ghost point value from the temperature interpolation. First removing it from the summation yields

$$T_M = \sum_{i=1}^{8} f_{T,i} T_i = f_{T,j} T_j + \sum_{i=1, i\neq j}^{8} f_{T,i} T_i \tag{3.115}$$

Then, from the adiabatic boundary condition of Equation (3.106) we have $T_j = T_G = T_M$, and substituting into Equation (3.115) yields

$$T_M = f_{T,j} T_M + \sum_{i=1, i\neq j}^{8} f_{T,i} T_i \tag{3.116}$$

This is an implicit equation for T_M, and can be solved for T_M as

$$\boxed{T_M = \sum_{i=1, i\neq j}^{8} \frac{f_{T,i}}{1 - f_{T,j}} T_i} \tag{3.117}$$

General Form of Interpolation

For the interpolation of velocity, Equations (3.111) and (3.114) can be represented by a single equation for generality:

$$\boxed{\phi_M = \sum_{i=1}^{8} \widehat{f}_{\phi,i} \phi_i + S_\phi} \tag{3.118}$$

If the primary ghost point is not one of the interpolation points, then the interpolation factors are $\widehat{f}_{\phi,i} = f_{\phi,i}$ for $i = 1, ..., 8$ and the source term $S_\phi = 0$. If the primary ghost point is one of the interpolation points, then the interpolation factors are $\widehat{f}_{\phi,i} = \frac{f_{\phi,i}}{1+f_{\phi,j}}$ for $i \neq j$ and $\widehat{f}_{\phi,i} = 0$ for $i = j$ and the source term $S_\phi = \frac{2f_{\phi,j}}{1+f_{\phi,j}} \phi_B$.

For the interpolation of temperature, Equations (3.111) and (3.117) can be represented

by a single equation for generality:

$$\boxed{T_M = \sum_{i=1}^{8} \widehat{f}_{T,i} T_i} \tag{3.119}$$

If the primary ghost point is not one of the interpolation points, then the interpolation factors are $\widehat{f}_{T,i} = f_{T,i}$ for $i = 1, ..., 8$. If the primary ghost point is one of the interpolation points, then the interpolation factors are $\widehat{f}_{T,i} = \frac{f_{T,i}}{1-f_{T,j}}$ for $i \neq j$ and $\widehat{f}_{T,i} = 0$ for $i = j$.

Time-Lagged Values used in Interpolation

The value of the primary ghost point is removed from the interpolation for the mirror point by creating an implicit relation as discussed above. However, neighboring ghost points may be included in the interpolation depending on the location of the mirror point. This introduces a time "lag" in the interpolation since the neighboring ghost values are from the previous time-level, whereas the fluid points used in the interpolation have been updated to the current time-level. Thus, the primary ghost point value ϕ_G being updated is a function of current fluid values ϕ_i^n and lagged neighboring ghost values ϕ_j^{n-1}:

$$\phi_G = \phi_G(\phi_i^n, \phi_j^{n-1}) \quad \text{where} \quad i \neq j \tag{3.120}$$

where this function is a combination of Equations (3.105) and (3.118) if updating velocity ghost points, or a combination of Equations (3.106) and (3.119) if updating temperature ghost points.

In order to mitigate the effect of the lag, Picard iteration is performed on Equation (3.120) to successively update the ghost points to advance them to the same time-level as the fluid points. During Picard iteration the fluid point values ϕ_i^n remain the same, whereas the ghost point values ϕ_j^{n-1} change. While the number of Picard iterations to achieve a

prescribed level of improvement was not explored, it was decided that three Picard iterations would be sufficient to improve the accuracy, at least beyond using just a single iteration.

3.6.3 Procedure for Ghost Cell Method in Navier-Stokes Solver

Preprocessing Steps

The following steps are for preprocessing and are performed only once on the CPU. Figure 3.14 shows the geometric relationship between the variables that will be used in the following steps.

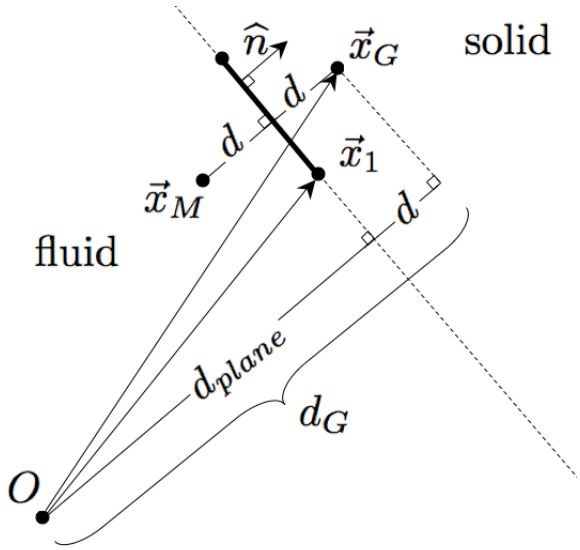


Figure 3.14: 2D view of a segment lying in a plane at a distance of d_{plane} from the global origin O.

1. Input vertices $\vec{x}_1, \vec{x}_2, \vec{x}_3$ for each triangular segment of immersed surface.

2. Compute unit normal vector for each segment:

$$\widehat{n} = \frac{\vec{x}_{12} \times \vec{x}_{13}}{|\vec{x}_{12} \times \vec{x}_{13}|} = \frac{\vec{n}}{|\vec{n}|} \tag{3.121}$$

The vertices are numbered clockwise around the segment when viewed from outside the obstacle. This ensures the normal points *inside* the obstacle, which is the convention adopted here.

3. Compute perpendicular distance from global origin to plane containing each segment:

$$d_{plane} = \widehat{n} \cdot \vec{x}_1 \tag{3.122}$$

We can pick any point in the plane of the segment, but as a convention we choose the $\vec{x}_1$ vertex of the segment.

Steps 4. to 9. are performed separately for velocity ghost points (cell face centers) and temperature ghost points (cell centers).

4. Determine the cell centers and face centers that lie inside obstacle. This is done by computing

$$d = \widehat{n} \cdot \vec{x} - d_{plane} \tag{3.123}$$

where d is the signed distance from the ghost point to the plane of the segment and $\vec{x}$ is the location for the x-face, y-face, z-face, or cell-center. Equation (3.123) is computed for each segment, and if $d > 0$ for all segments, then the point $\vec{x}$ lies inside the obstacle. These points are tagged with a 0 (interior points), and the remaining points are tagged with a 1 (fluid points).

5. Determine the interior points that are adjacent to immersed boundary by searching for a fluid point in all six directions. If a neighboring fluid point is found, then the interior point is assigned as a ghost point, tagged with a 2.

6. Determine the closest segment for each ghost point. This is done by finding the minimum perpendicular distance from the ghost point to all the segments.

7. Compute the mirror points by reflecting each ghost point location across its closest

segment along the normal:

$$\vec{x}_M = \vec{x}_G - 2d\widehat{n} \tag{3.124}$$

where d is the magnitude of the perpendicular distance from the ghost point to the plane containing the segment:

$$d = |d_G - d_{plane}| = |\widehat{n} \cdot \vec{x}_G - d_{plane}| \tag{3.125}$$

8. A search is performed in the mesh to determine the cell that the mirror point lies in, and then the nearest-neighbor cells are determined. The eight velocity (or temperature) points corresponding to these eight cells are used as interpolation points for the mirror point. The locations of the eight interpolation cells are stored for later use.

9. The interpolation factors ($\widehat{f}_{u,i}$, $\widehat{f}_{v,i}$, $\widehat{f}_{w,i}$, or $\widehat{f}_{T,i}$ for $i = 1, ..., 8$) for each mirror point are computed and stored.

Steps during Time Integration

The following procedure is performed for each time-step on the GPU.

1. Update ghost points. The ghost points are updated by first interpolating to get the mirror point value. The interpolation factors and interpolation point locations that were stored during the preprocessing steps above are now used. Then a boundary condition is applied at the segment, which sets the ghost point value. If interpolating for velocity, the mirror point value is found from Equation (3.118) and the ghost point value is found from the no-slip condition:

$$\phi_G = 2\phi_B - \phi_M \tag{3.126}$$

If interpolating for temperature, the mirror point value is found from Equation (3.119)

and the ghost point value is found from the adiabatic boundary condition:

$$T_G = T_M \tag{3.127}$$

This step is implemented on a GPU by having separate kernels for u, v, w, and T. The number of threads used in each kernel is equal to the number of ghost points. A read/write conflict on the GPU is introduced since the interpolation can potentially include a neighboring ghost point, and it is uncertain if the neighboring ghost point has already been updated by its assigned thread. This is remedied by using separate storage for the updated ghost point values. When kernel execution ends all ghost points have been updated and stored separately. Then another kernel is launched which updates the main variable arrays using the updated ghost point values that were stored separately. Both kernels are nested within a Picard iteration loop to successively update the ghost values to improve accuracy.

2. Update temperature fluid points using the energy equation.

3. Compute provisional velocity at fluid points. This provisional velocity is not divergence-free.

4. Calculate mass residual. The mass residual for the cell surrounding the pressure ghost points excludes the ghost velocity values, since the ghost velocities are not solutions of the Navier-Stokes equations. Instead, the boundary velocity (which is zero for a stationary boundary) is directly substituted in the mass residual calculation for cell faces that contain a ghost velocity. Thus, the mass residual is calculated using a "stair-step" representation of the boundary.

5. Calculate pressure. The pressure ghost points are updated in the same way as the fluid points, using the pressure-Poission equation. The pressure coefficients for cell faces containing a ghost velocity are set to zero. The present method ensures local

mass conservation in the ghost cells, global mass conservation over the entire mesh, and no mass flux through the immersed boundary.

6. Update velocity at fluid points using pressure.

Chapter 4

Validation Cases

4.1 Introduction

The chapter presents cases used to validate the flow solver. The cases include canonical flows with well established physics, so that validation would be unambiguous. Three cases are presented: simulation of laminar flow in a 3D driven cavity, DNS of turbulent flow in a square duct, and DNS of turbulent flow in a circular pipe. The results of the present simulations are compared with data from the literature. In addition, computational performance of the single-GPU and multi-GPU solvers will be assessed.

4.2 Laminar Flow in a 3D Lid-Driven Cavity

The first validation problem is the laminar flow in a lid-driven cavity. The computational domain is shown in Figure 4.1. The Reynolds number based on the lid speed u_{lid} and edge length L was $Re_L = u_{lid}L/\nu = 1000$. The top lid at $z = L$ is moving with velocity $u = u_{lid}$ and the other walls are stationary. The cube domain was meshed using 128 x 128 x 128 cells with constant mesh spacing.

The solution was advanced until a steady-state was achieved using a time-step of $\Delta t = 0.001(L/u_{lid})$. The resulting velocity profiles along the vertical centerline ($x = L/2, y = L/2, 0 \leq z \leq L$) and horizontal centerline ($y = L/2, z = L/2, 0 \leq x \leq L$) are shown in Figure 4.2 and are compared to the results of Ku et al. [103]. This simulation was performed using the single GPU solver and multi-GPU solver using four GPUs in parallel.

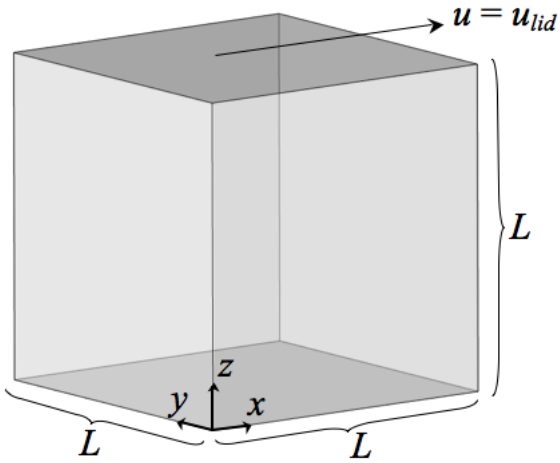
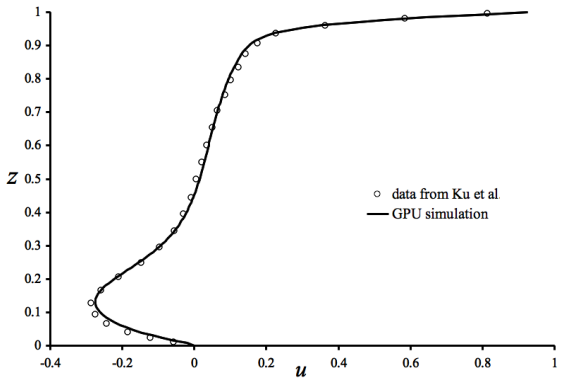
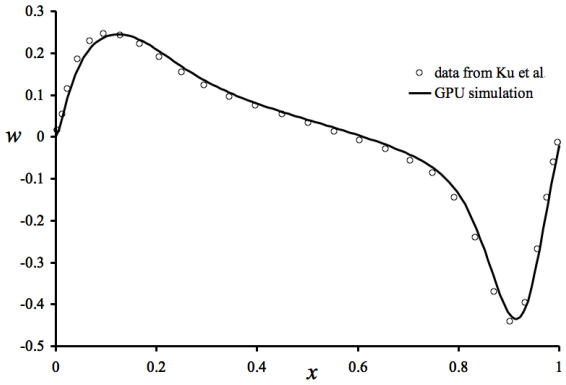


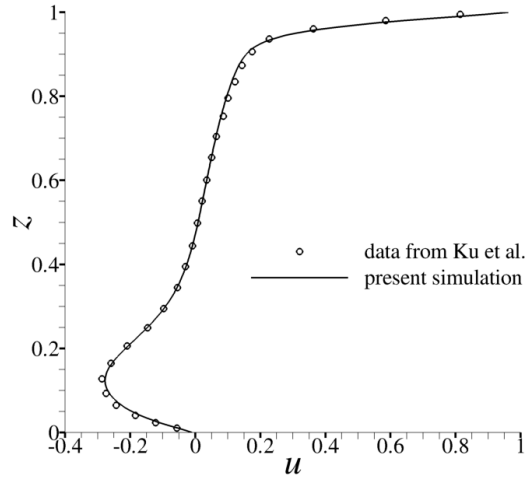
Figure 4.1: Computational domain for lid-driven cavity.



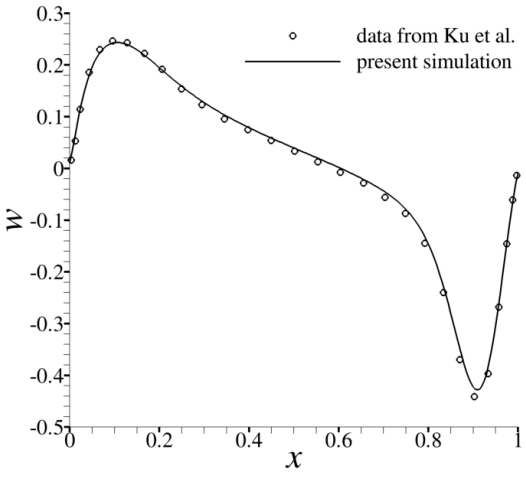
(a) single-GPU solver



(b) single-GPU solver



(c) multi-GPU solver



(d) multi-GPU solver

Figure 4.2: Lid-driven cavity simulation using single-GPU and multi-GPU solver: (a), (c) u velocity along vertical centerline and (b), (d) w velocity along the horizontal centerline. Solid line: present simulation, symbols: data from Ku et al. [103].

The simulation results for both the single and multi-GPU solver are in excellent agreement with [103], thus providing validation of the flow solver for laminar flows.

4.3 DNS of Turbulent Flow in a Square Duct

A direct numerical simulation of turbulent flow in a square duct at $Re_\tau = 360$ (bulk Reynolds number of 5480) was performed. The computational domain is shown in Figure 4.3. The characteristic length is the hydraulic diameter D_h and the characteristic velocity is the mean friction velocity u_τ. The hydraulic diameter is defined as $D_h = 4(area)/(perimeter)$ and since the width and height of the cross-section is D we have $D_h = D$. The flow is driven by a constant mean pressure gradient added as a source term to the momentum equation: $\mathbf{F} = -\frac{d\overline{p}^*}{dx^*}\mathbf{i} = 4\mathbf{i}$. No-slip boundary conditions for the velocity were applied to the solid duct walls, and a periodic boundary condition on velocity was applied to the inlet and outlet of the duct. The dimensionless lengths of the domain were $D/D_h = 1$ and $L/D_h = 8$. Previous studies [104, 105] have found that this streamwise length is sufficient to ensure that the turbulent structures are sufficiently uncorrelated, which is necessary to ensure that the streamwise periodic boundary condition is applicable.

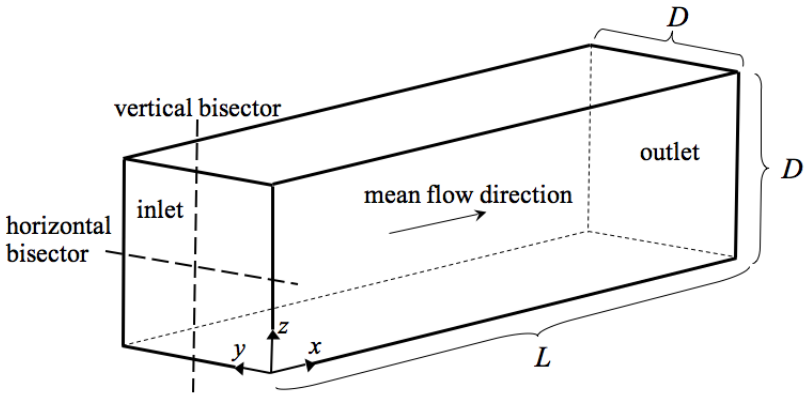


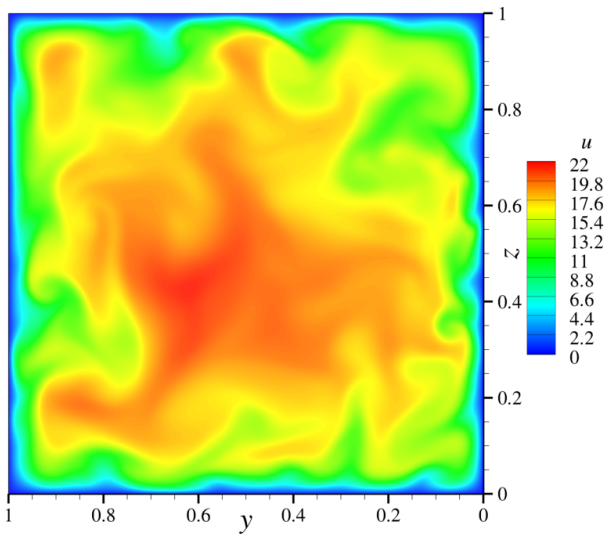
Figure 4.3: Computational domain for square duct.

The mesh had 1024 cells in the streamwise direction, 160 cells in the spanwise direction, and 160 cells in the vertical direction. This gives approximately 26.2 million cells in the

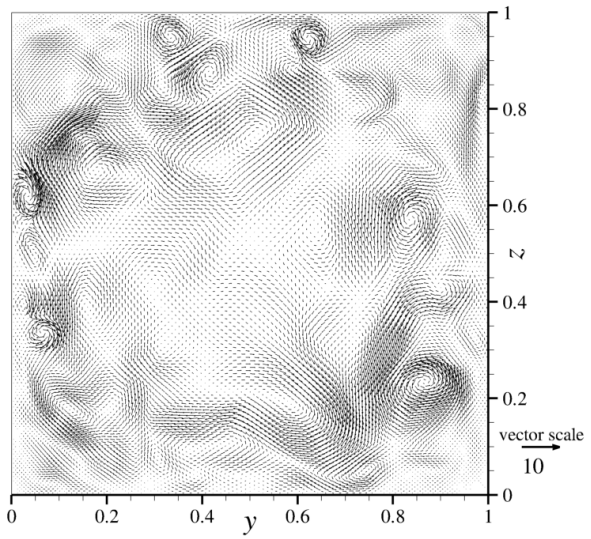
mesh. The mesh cells were uniform in the streamwise x-direction and geometric stretching away from the walls of 1 percent was used in the cross-flow (y-z) plane. This validation case was used not only to test the predictive capability of the flow solver, but also to test the limits of the GPU, since for the present implementation a mesh size of 26.2 million cells was the maximum mesh size that could be accommodated on a single Tesla GPU. Using a time-step of $\Delta t = 0.0001(D_h/u_\tau)$, the simulation was integrated for $50(D_h/u_\tau)$ time units. The mean statistics were collected starting at $5(D_h/u_\tau)$ and the root-mean-squares and Reynolds stresses were collected starting at $10(D_h/u_\tau)$. A sinusoidal perturbation was used to initiate turbulence, since the initial condition was a uniform velocity field. This perturbation was only used for the first $0.15(D_h/u_\tau)$ time units (or first 1500 time-steps).

The instantaneous flow at a cross-flow plane at $x/D_h = 4$ is shown in Figure 4.4, where the mean flow is directed into the figure. This realization of the turbulent flow was taken at the end of the simulation. Contours of the instantaneous streamwise velocity are shown in Figure 4.4(a), which highlight ejections of fluid from the wall in the shape of mushroom-like structures. One such structure is located on the left wall, where the stem originates at around $y/D_h = 1$, $z/D_h = 0.35$. A larger mushroom ejection occurs at the right wall, where the stem originates at around $y/D_h = 0$, $z/D_h = 0.7$. Cross-flow vectors with components v/u_τ and w/u_τ are shown in Figure 4.4(b), which highlight the instantaneous eddy structures in the flow.

The mean flowfield in the cross-flow plane of the duct is shown in Figure 4.5, which was obtained by streamwise averaging every plane of data for the mean flowfield to obtain a single plane representing the mean flow. In Figure 4.5(a) the vectors indicate a secondary flow pattern, where each quadrant of the duct has a pair counter-rotating vortices; this secondary flow is known as "secondary flows of Prandtl's second kind." The secondary velocity was approximately 2 percent of the bulk streamwise velocity, indicating that the time-averaged secondary flow is small. A closer examination of the secondary flow in the quadrant nearest the origin is shown in Figure 4.5(b). It can be seen that the vortices are

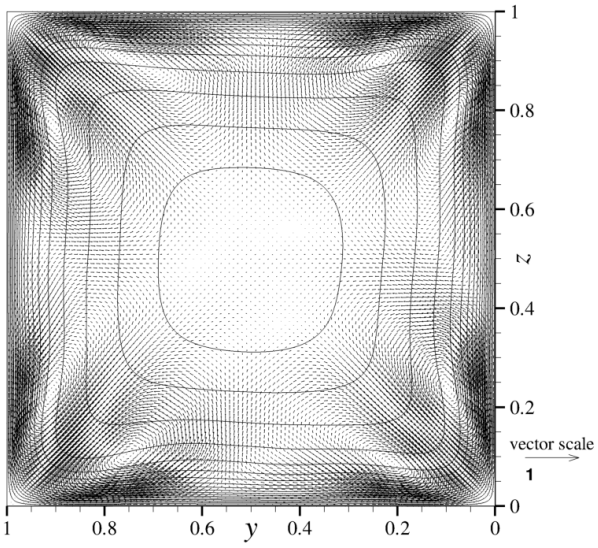


(a) Contours of u/u_τ

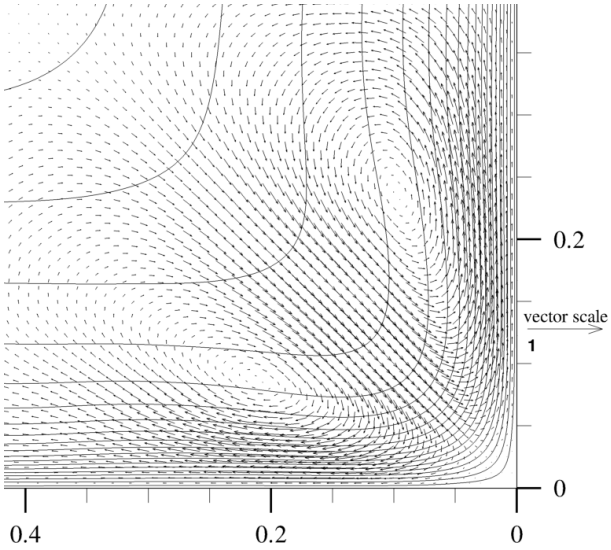


(b) Cross-flow vectors with components $(v/u_\tau, w/u_\tau)$

Figure 4.4: Instantaneous flow field at $x/D_h = 4$.



(a) Full cross-section



(b) Lower-right corner

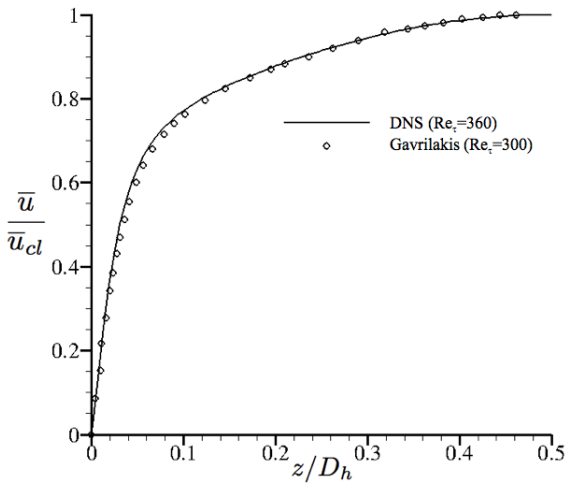
Figure 4.5: Cross-flow vectors of mean secondary flow and contour lines of mean streamwise velocity. Mean flow is directed into the paper.

symmetric about the corner bisector and that fluid moves from the core of the duct toward the corners. The contour lines represent the non-dimensional mean streamwise velocity, which is the dominant component of the mean flow, directed into the figure. A total of 19 contour lines are shown, where the minimum contour has a value equal to 1 (located nearest the wall), the maximum contour has a value of 19 (near the duct center), and the contour increment is unity. The maximum value for $\overline{u}/u_\tau$ is approximately 19.97 at the duct centerline. The bulging of the contour lines in the vicinity of each corner is caused by the transfer of momentum toward the corner by the secondary flow.

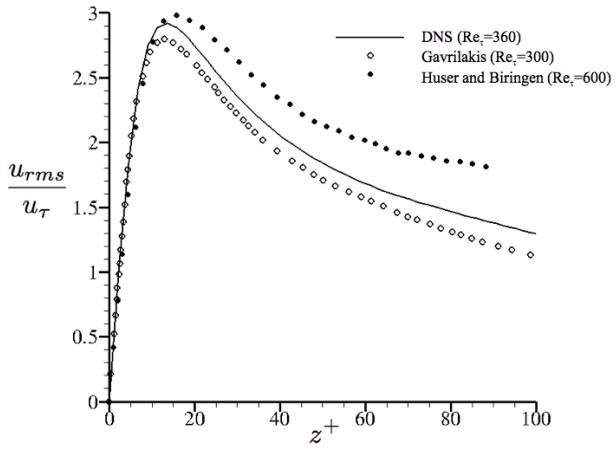
The profile of the mean streamwise velocity is shown in Figure 4.6(a), where the profile was measured along the vertical bisector of the duct and is normalized by the mean streamwise velocity at the duct centerline. The expected blunt velocity profile is observed, typical of turbulent flow. The profile is plotted only for half the distance along the bisector, since the profile is symmetric about the duct centerline. The present profile is compared to the DNS of Gavrilakis [106] and good agreement is observed. Figure 4.6(b) shows a profile of the root-mean-square of the streamwise velocity fluctuations along the vertical bisector of the duct. The data are plotted in terms of wall units in the near-wall region. The present data and that of Gavrilakis [106] and Huser and Biringen [105] collapse on the same curve until near the peak, after which the agreement is reasonable considering the difference in Reynolds numbers.

4.4 DNS of Turbulent Flow in a Circular Pipe

Another validation problem is the DNS of fully-developed turbulent flow in a circular pipe. The pipe diameter is d and domain length of the pipe was taken as $5d$. The characteristic length is pipe diameter d and the characteristic velocity is the mean friction velocity u_τ. The friction Reynolds number was set as $Re_\tau = u_\tau d/\nu = 340$, which corresponds to a bulk Reynolds number of $Re_b = 5000$ based on the bulk velocity u_b. The flow was driven by



(a) Mean streamwise velocity



(b) Root-mean-square of streamwise velocity fluctuations ($z^+ = Re_\tau(z/D_h)$)

Figure 4.6: Statistics along vertical bisector of square duct compared to data from Gavrilakis [106] and Huser and Biringen [105].

a constant mean pressure gradient in the streamwise direction, which is added as a source term to the momentum equation. This dimensionless source term can be derived from the fact that the mean pressure gradient must balance the net viscous friction at the pipe wall [107], which yields

$$-\frac{d\overline{p}}{dx} = \frac{4u_\tau^2}{d} \tag{4.1}$$

or non-dimensionally

$$-\frac{d\overline{p}^*}{dx^*} = 4 \tag{4.2}$$

Thus the dimensionless source term added to the momentum equation is

$$\mathbf{F} = -\frac{d\overline{p}^*}{dx^*}\mathbf{i} = 4\mathbf{i} \tag{4.3}$$

The mesh had 128 cells along the diameter and 512 cells in the streamwise direction. No-slip boundary conditions were applied at the walls and a periodic boundary condition was used in the streamwise direction. A time-step of $\Delta t = 0.00015(d/u_\tau)$ was used, and the simulation was integrated for $t = 90(d/u_\tau)$ time units. Mean statistics were collected starting at

$t = 15(d/u_\tau)$ time units and root-mean-square statistics and Reynolds shear stresses were collected starting at $t = 30(d/u_\tau)$ time units. Figure 4.7 is a cross-sectional view of the pipe at $x/d = 2.5$ showing contours of instantaneous streamwise velocity u/u_τ and cross-flow vectors with components v/u_τ and w/u_τ. The flow in the cross-flow plane of the pipe shows turbulent eddies, indicating strong mixing in the fluid. Figure 4.8 shows the mean streamwise velocity profile as a function of pipe radius. The expected blunt velocity profile is observed and is compared to the result obtained by Muppidi and Mahesh [16], with excellent agreement. In addition, results for the root-mean-square statistics and Reynolds shear stress are plotted in Figure 4.9, with good agreement versus the statistics of [16].

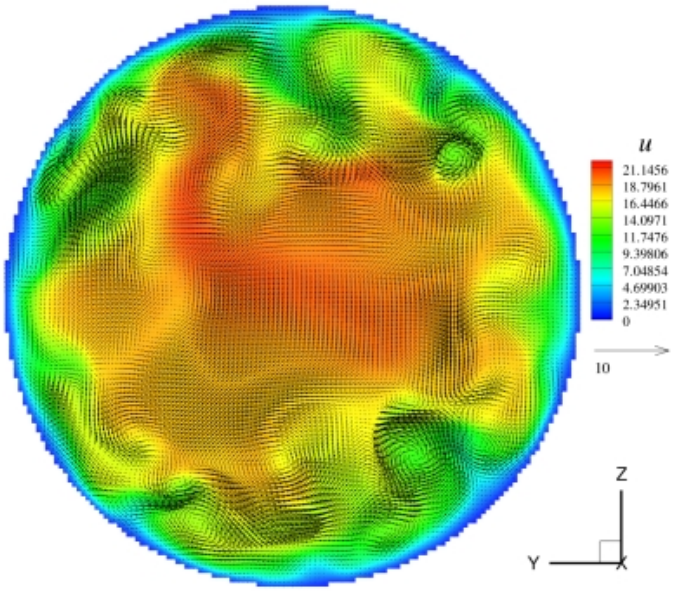


Figure 4.7: Turbulent pipe flow at $Re_b = 5000$: Instantaneous streamwise velocity u/u_τ with cross-flow vectors superimposed at $x/d = 2.5$.

4.5 Performance of GPU-based flow solver

The performance of the GPU-based Navier-Stokes solver will now be assessed. We compare the single-GPU solver to a comparable implementation in Fortran, and also compare the single-GPU and multi-GPU solver. The CUDA code was compiled and executed using the

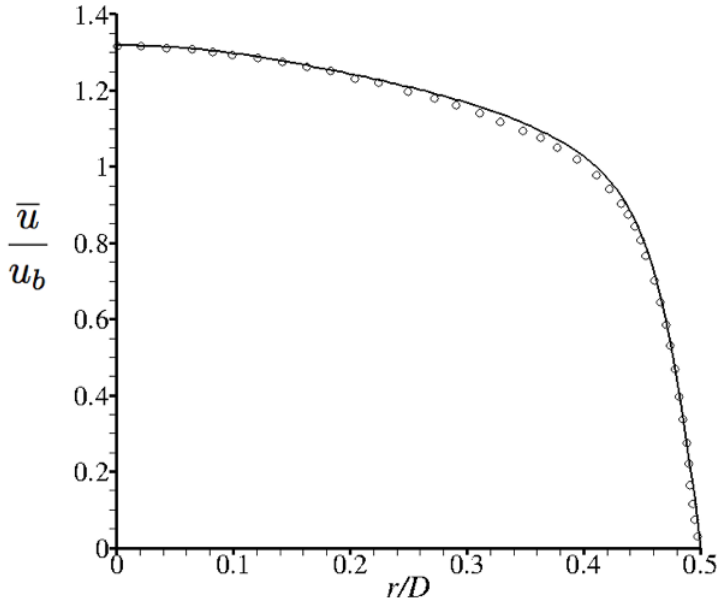
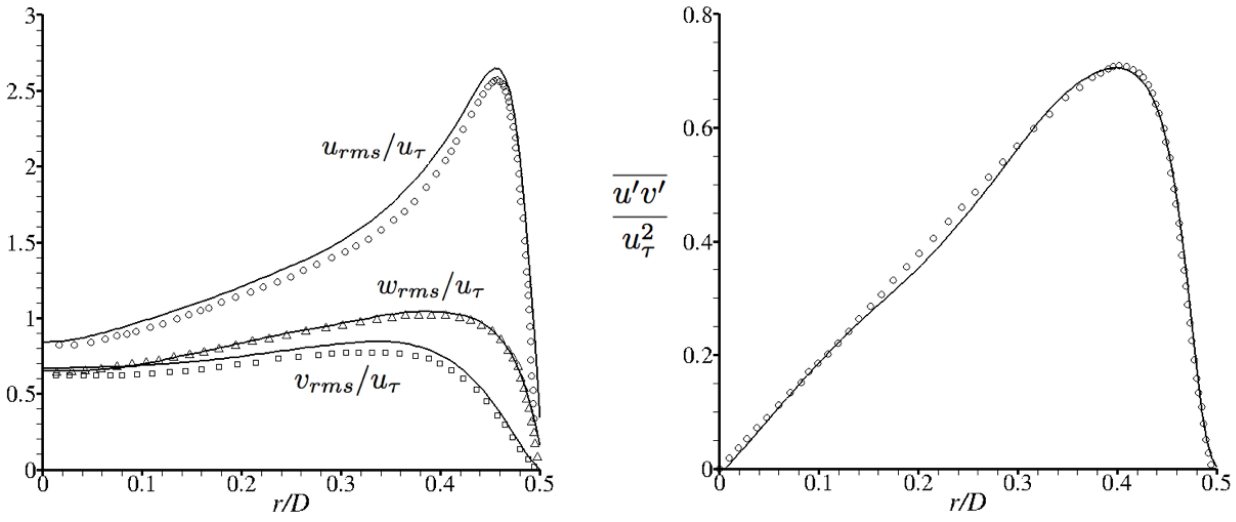


Figure 4.8: Turbulent pipe flow at $Re_b = 5000$: Mean streamwise velocity profile as a function of pipe radius, where the line is from the present DNS and symbols are data from Muppidi and Mahesh [16].



(a) Root-mean-square statistics

(b) Reynolds shear stress

Figure 4.9: Turbulent pipe flow at $Re_b = 5000$: Root-mean-square statistics (a) and Reynolds shear stress (b), where the line is from the present DNS and symbols are data from Muppidi and Mahesh [16].

nvcc compiler and the Fortran code was compiled using the gfortran compiler with -02 optimization. Both the GPU and CPU codes used single-precision. For the GPU-to-CPU comparisons, a single NVIDIA Tesla C1060 GPU is compared to a single core of a 2.6 GHz AMD Phenom quad-core processor. For the multi-GPU comparison, all GPUs used were the NVIDIA Tesla C1060. All tests were performed using the same 64-bit Linux workstation.

The lid-driven cavity simulation was used to assess performance of the single-GPU solver versus an equivalent CPU-based version of this code written in Fortran. The two codes were timed for the first 100 time-steps of simulation for various mesh resolutions; the results are shown in Table 4.1. It was found that the speedup increases with mesh resolution, and that for reasonable size meshes the GPU solver provided over an order-of-magnitude speedup compared to a CPU. This trend is logical, since for larger problems more threads must be used and the efficiency of the GPU is realized. The single-GPU solver was also tested for the square duct simulation, and a comparison of the performance for the first 100 time-steps for various mesh resolutions is shown in Table 4.2. The speed-up increases with mesh size until the last mesh, where the speedup decreases. The lid-driven cavity was also used to assess the performance of the multi-GPU solver relative to the single-GPU version. These results are shown in Table 4.3, and it was found that the multi-GPU solver delivers sub-linear speed-up scaling, as expected due to the communication overhead. Further optimization of both the single and multi-GPU solvers will yield better speedup.

Table 4.1: Comparison of performance for CPU code (Fortran) versus GPU code (CUDA) for laminar flow in lid-driven cavity. Timings taken for first 100 time-steps of simulation.

mesh	CPU time (sec)	GPU time (sec)	speedup (CPU time/GPU time)
16x16x16	0.46	0.39	1.18
32x32x32	4.49	0.83	5.41
64x64x64	46.15	3.35	13.78
128x128x128	420.20	22.40	18.76

Table 4.2: Comparison of performance for CPU code (Fortran) versus GPU code (CUDA) for turbulent flow in a square duct. Timings taken for first 100 time-steps of simulation.

mesh	CPU time (sec)	GPU time (sec)	speedup (CPU time/GPU time)
128x32x32	27.63	2.30	12.02
256x64x64	275.96	15.39	17.93
512x64x64	569.04	30.45	18.69
512x128x128	1997.05	126.06	15.84

Table 4.3: Multi-GPU performance for laminar flow in 3D lid-driven cavity. Timings (in seconds) taken for first 100 time-steps of simulation.

mesh	1 GPU	2 GPUs	speedup	4 GPUs	speedup
32x32x32	1.70	1.23	1.38	0.99	1.71
64x64x64	4.76	3.08	1.55	2.18	2.18
128x128x128	27.12	15.53	1.75	9.92	2.73
256x256x256	214.56	113.91	1.88	64.86	3.31

4.6 Conclusions

In this chapter, three validation cases were presented: simulation of laminar flow in a 3D driven cavity, DNS of turbulent flow in a square duct, and DNS of turbulent flow in a circular pipe. The driven cavity simulation compared very well versus previous data, indicating the solver is capable of simulating laminar flows. The square duct simulation correctly predicted the expected qualitative turbulent features, including flow ejection from the wall in the instantaneous flow and secondary flow vortices in the mean flow. Comparison with previous statistics from the literature indicated the present predictions were reasonable. The pipe flow simulation demonstrated that the anisotropy in the root-mean-square statistics could be correctly captured, and that the Reynolds shear stress was correctly predicted. The successful simulation of the preceding flows provides confidence in the present GPU-based CFD solver for simulating flows in basic geometries. In addition, the assessment of GPU performance indicated the solver is, on average, an order of magnitude faster than a single CPU. Some areas of improvement that can enhance performance are better data structure

layouts to improve memory coalescing, decreased branch divergence by removing conditional statements where possible, and improved usage of shared memory.

Chapter 5

LES of Film-Cooling Flow with a Micro-Ramp Vortex Generator: Jet Modeled with Precursor Simulation

5.1 Introduction

Large Eddy Simulations are performed to study an inclined turbulent jet interacting with a cross-flow in a film-cooling configuration. The purpose is to compare how a micro-ramp vortex generator placed downstream of the jet alters the flowfield and heat transfer characteristics compared to a baseline case with no micro-ramp. A total of four simulations have been performed: a baseline flow with no micro-ramp using coarse and fine meshes, and a flow including a micro-ramp using coarse and fine meshes. Only the flowfield results obtained using the fine mesh will be shown, and the results obtained using the coarse mesh will only be used to assess the effects of mesh resolution on the prediction of film-cooling effectiveness. The baseline flow was based upon the DNS study of Muldoon and Acharya [19] to provide a means for comparison. In both the present study of this chapter and the previous work of [19], the jet exit conditions were prescribed using a precursor simulation. Note that this is in contrast to the configuration used in the next chapter, where the jet exit conditions were created by including a plenum and a jet pipe in the computational domain.

This chapter is organized as follows: first, the problem description is given along with the numerical methods and boundary conditions. Next, the results are presented which include instantaneous and time-averaged results for the velocity and temperature fields. Coherent turbulent structures characteristic of a jet in a cross-flow are identified and discussed. Results for film-cooling effectiveness will be shown to quantify the improvement delivered by the micro-ramp, compare with previous DNS results from the literature, and to assess the effect

of mesh resolution.

5.2 Problem Description

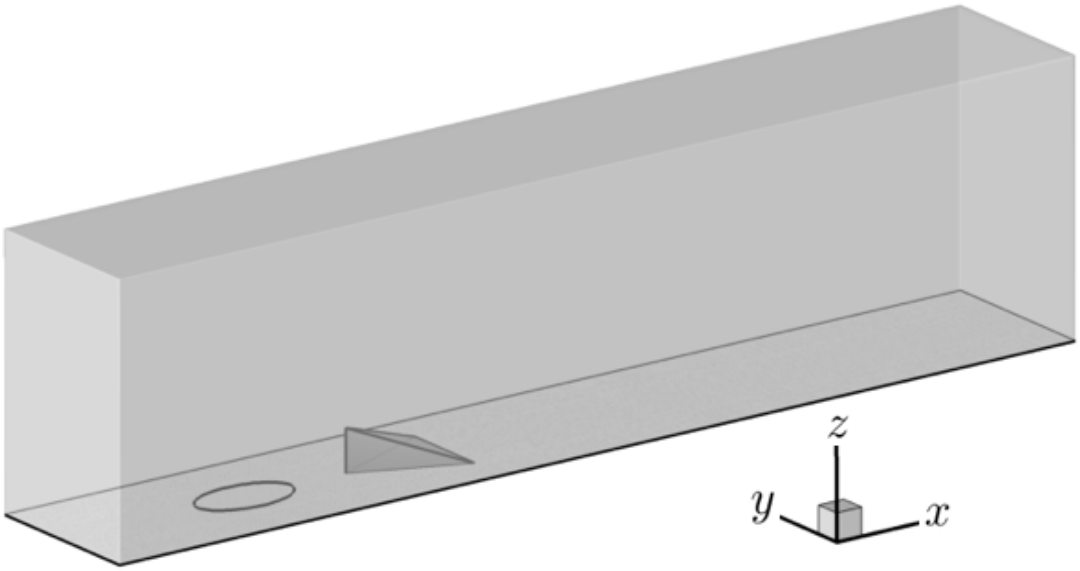
Instead of simulating the film-cooling flow over an actual turbine blade geometry, a simplified geometry is used to extract the fundamental flow physics. The computational domain is shown in Figure 5.1 and the corresponding dimensions are given in Table 5.1.

Table 5.1: Dimensions of computational domain.

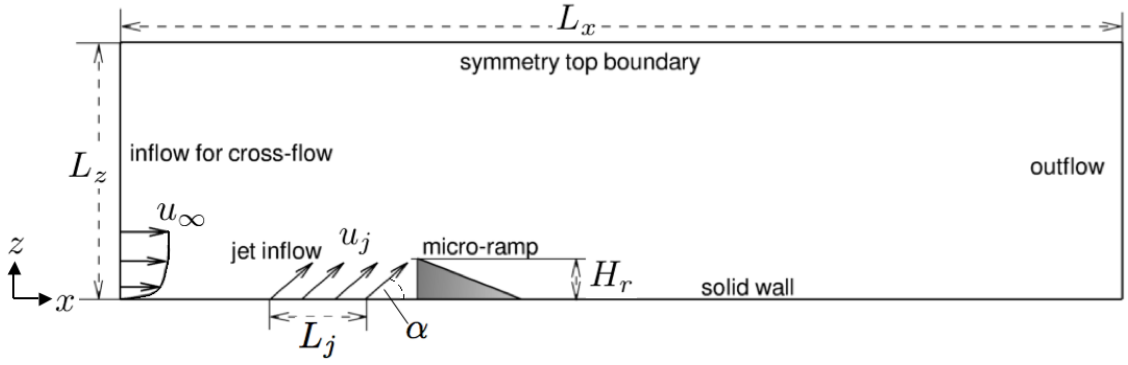
L_x	$18.4d$
L_y	$3.0d$
L_z	$4.7d$
L_{LE}	$2.7d$
L_j	$1.743d$
L_{jr}	$1.0d$
L_r	$1.913d$
S_r	$1.573d$
H_r	$0.748d$
α	35 degrees

Coolant is injected into the cross-flow at an angle of 35 degrees to the freestream. The blowing ratio (u_j/u_∞) was 1.5 and the Reynolds number based on the jet diameter and freestream cross-flow velocity was $Re = u_\infty d/\nu = 8,000$. This problem was constructed to be similar to that used by Muldoon and Acharya [19] in order to validate the baseline simulation (no micro-ramp) against their results.

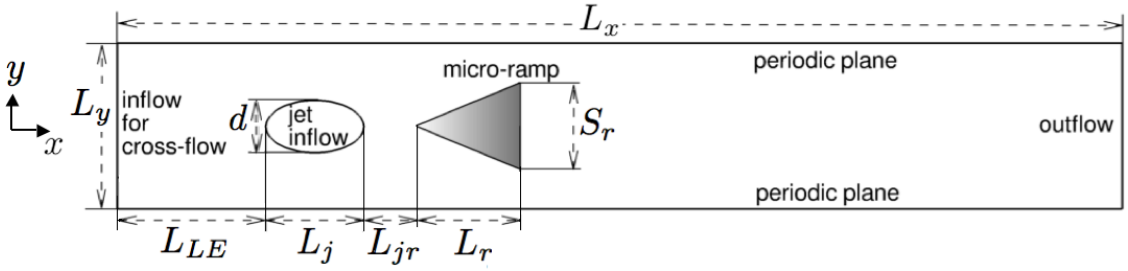
In both the present study and in the previous work of Muldoon and Acharya [19], the coolant delivery pipe and plenum are not included in the simulation. Instead, the jet inflow velocity boundary condition was prescribed using a precursor simulation. By prescribing the velocity boundary condition in this way implies that the jet inflow is not allowed to adjust to the effect of the cross-flow, which may change the results. However, this approximation has allowed higher resolution above the flat plate to resolve the turbulent scales in the jet/cross-



(a) three-dimensional view



(b) side view



(c) top view

Figure 5.1: Computational domain for film-cooling configuration with a micro-ramp.

flow interaction region. This is especially important since the downstream flow structures and scalar transport are the primary focus. In the present approach, the jet exit boundary condition was constructed using the instantaneous velocity field from a precursor simulation, as described in section 5.4.6. This differs from that used in [19], where they applied mean velocities for the jet inflow based on a precursor simulation. Our present method is believed to be more realistic, since it supplies instantaneous velocity data at the inflow, which is necessary for LES.

The micro-ramp used in the present simulation is based on the H0(R0) model used in the experimental study by Zaman et al. [23]. However, these experiments were at a higher Reynolds number than the present LES, and hence were not used for direct quantitative comparison. However, this same micro-ramp geometry is used in the next chapter for simulations at the experimental conditions, which will be compared to the experimental results in [23].

5.3 Numerical Methodology

Large Eddy Simulations are performed using the GPU-based flow solver CU-FLOW, which was described in Chapter 3. The WALE eddy viscosity model was used with the model constant of $C_w = 0.55$. The LES equations were non-dimensionalized using the jet exit diameter d and the freestream velocity u_∞. Thus the Reynolds number is $Re_\infty = u_\infty d/\nu$. The non-dimensional temperature is defined as $T^* = (T - T_j)/(T_\infty - T_j)$, where T_∞ is the freestream temperature of the cross-flow (assumed constant) and T_j is the jet temperature (assumed constant). The temperature is one-way coupled with the velocity field and is treated as a passive conserved scalar. The Prandtl number is taken as unity, and thus the momentum and energy equations have the same diffusion coefficient.

A Cartesian mesh with non-uniform spacing was used for the LES. Simulations for the baseline case and the micro-ramp case were performed using coarse and fine meshes to assess

the effects of mesh resolution on film-cooling predictions. The fine mesh resolution was 800 cells (streamwise) x 160 cells (spanwise) x 160 cells (vertical) $\approx$ 20.5 million cells, and the coarse mesh resolution was 512 cells (streamwise) x 128 cells (spanwise) x 128 cells (vertical) $\approx$ 8.4 million cells. For the fine mesh, the jet inflow area was meshed using 128 cells along the major axis and 64 cells along the minor axis; for the coarse mesh the jet inflow area was meshed using 64 cells along the major axis and 64 cells along the minor axis. The mesh spacings in the jet inflow area were uniform such that $\Delta x_j = L_j/128$ (fine mesh) or $\Delta x_j = L_j/64$ (coarse mesh) and $\Delta y_j = d/64$. The streamwise and spanwise mesh spacings were set by stretching geometrically from the jet inflow area. A boundary layer mesh was generated at the wall; for the fine mesh, this included 70 cells inside of a vertical height of $\delta = d$, where the first cell at the wall had a spacing of $\Delta z/d \approx 0.01$, and for the coarse mesh included 37 cells inside of a vertical height of $\delta = d$, where the first cell at the wall had a spacing of $\Delta z/d \approx 0.02$. Since a Cartesian mesh was used, the surfaces of the micro-ramp were not boundary-fitted by the mesh. A ghost cell immersed boundary method (see Chapter 3) was used to ensure that the Dirchlet and Neumann boundary conditions were exactly satisfied at the micro-ramp surfaces. Since the micro-ramp geometry is composed of three triangles, the surface can be represented in the ghost cell method using just three segments that are coincident with the micro-ramp faces.

The time-step used was $\Delta t = 0.001(d/u_\infty)$, which was selected to ensure temporal accuracy and satisfy the CFL condition, which is a restriction imposed by the explicit nature of the algorithm. The problem was simulated for a total of $400(d/u_\infty)$ time units, and mean statistics were collected starting at $20(d/u_\infty)$ time units and continued until the end of the simulation. Using a Tesla C1060 GPU, the LES of the flow with no micro-ramp took 8.0 $seconds/\Delta t$ on the fine mesh and the LES of the flow with a micro-ramp took 11.4 $seconds/\Delta t$ on the fine mesh.

5.4 Boundary Conditions

5.4.1 Inflow Boundary

At the inflow boundary, the streamwise u velocity is prescribed as a 1/7th power law

$$u^* = \frac{u}{u_\infty} = \left(\frac{z}{\delta}\right)^{1/7} \tag{5.1}$$

with boundary layer height of one diameter ($\delta = d$). The cross-flow components were set as zero ($v^* = w^* = 0$). The inflow temperature was prescribed as T_∞ (non-dimensionally $T^* = 1$) to represent the hot cross-flow.

5.4.2 Top Boundary

At the top boundary, a symmetry boundary condition is used, representing a frictionless wall. For the velocities this is written as

$$\frac{\partial u^*}{\partial z^*} = \frac{\partial v^*}{\partial z^*} = w^* = 0 \tag{5.2}$$

and for the temperature as

$$\frac{\partial T^*}{\partial z^*} = 0 \tag{5.3}$$

where the gradient was implemented numerically using a first-order accurate one-sided difference.

5.4.3 Side Boundaries

For the side boundaries, a periodic boundary condition was used. This simulates the effect of having a spanwise row of film-cooling jets that are parallel to one another, where the jet centers are separated by a distance of $L_y = 3d$, which is a common spacing for jets of this type. This condition was applied to all variables in the solution (u, v, w, T, p).

5.4.4 Solid Boundaries

At solid boundaries (bottom flat plate and micro-ramp surface) the no-slip condition is applied for velocity,

$$u^* = v^* = w^* = 0 \tag{5.4}$$

and an adiabatic boundary condition is used for temperature,

$$\frac{\partial T^*}{\partial n} = 0 \tag{5.5}$$

where n is the local wall-normal coordinate. In the case of the flat plate wall $n = z^*$ and for the micro-ramp surface n is along the surface normal of each triangular segment.

5.4.5 Outflow Boundary

At the outflow boundary a convective boundary condition is used on all velocity components and temperature:

$$\frac{\partial \phi}{\partial t} + u_c \frac{\partial \phi}{\partial x} = 0 \tag{5.6}$$

where ϕ is taken as $u,\ v,\ w$ or T. The convective velocity u_c is taken as a constant and is equal to the freestream velocity u_∞ [108, 109]. Equation (5.6) is implemented numerically using the upwind scheme:

$$\phi^{n+1}_{imax} = \phi^n_{imax} - \frac{u_c \Delta t}{(\Delta x)_{imax}} (\phi^n_{imax} - \phi^n_{imax-1}) \tag{5.7}$$

where the $imax$ index is the ith location for the cell whose components represent the outflow boundary conditions ($imax = nx + 1$ for u and $imax = nx + 2$ for $v,\ w,\ T$ since the mesh is staggered). The grid spacing $(\Delta x)_{imax}$ is the spacing between ϕ^n_{imax} and ϕ^n_{imax-1}. Note that for stability the upwind scheme is subject to the CFL condition: $u_c \Delta t / (\Delta x)_{imax} < 1$.

The velocity component normal to the outflow boundary computed from Equation (5.7)

is scaled to ensure mass is conserved in the entire computational domain. In the present simulations this component is the u-velocity, which we call $\widehat{u}_{imax}$ to represent it as a provisional value. We then correct it using the ratio of the total volume flow rate into the domain to the volume flow rate through the outflow boundary,

$$u_{imax}^{n+1} = \widehat{u}_{imax} \frac{Q_{in}}{Q_{out}} \tag{5.8}$$

Lastly, note that non-dimensionalizing Equation (5.6) using d and u_∞ (and also using $(T - T_j)/(T_\infty - T_j)$ if $\phi = T$) yields

$$\frac{\partial \phi^*}{\partial t^*} + \frac{u_c}{u_\infty} \frac{\partial \phi^*}{\partial x^*} = 0 \tag{5.9}$$

The numerical discretization of this non-dimensional form is analogous to the previous method.

5.4.6 Jet Inflow Boundary Condition

Since the plenum and coolant pipe were not modeled in the present work, the jet inflow velocity boundary condition was created from a library of data from a precursor simulation. In an LES, the instantaneous velocities must be prescribed at the inflow, and should be done so as accurately as possible. Thus, instead of applying a steady boundary condition (such as a mean profile) for the jet inflow, it was decided that the instantaneous flowfield data from the precursor simulation would supply the boundary condition. In this way, the unsteadiness of the jet inflow would be better represented. The precursor simulation was an LES of turbulent pipe flow, which is used to represent the flow in the coolant delivery pipe in the film-cooling flow. Velocity data were extracted from the pipe simulation as if the pipe were connected to the jet inflow. This was done by passing an oblique plane at 35 degrees to the pipe axis through the pipe domain and saving velocity data along this plane. Figure 5.2 compares the domains of the precursor and film-cooling simulations. The

precursor domain is oriented in this figure such that the oblique plane through the pipe is parallel to the jet inflow. The two domains are pictured in this way to illustrate the point that data along the oblique plane passing through the pipe are mapped onto the jet inflow area of the film-cooling simulation. Note that the precursor mesh is not actually generated at this angle, rather it is simply shown at this angle to illustrate how the two domains relate to each other.

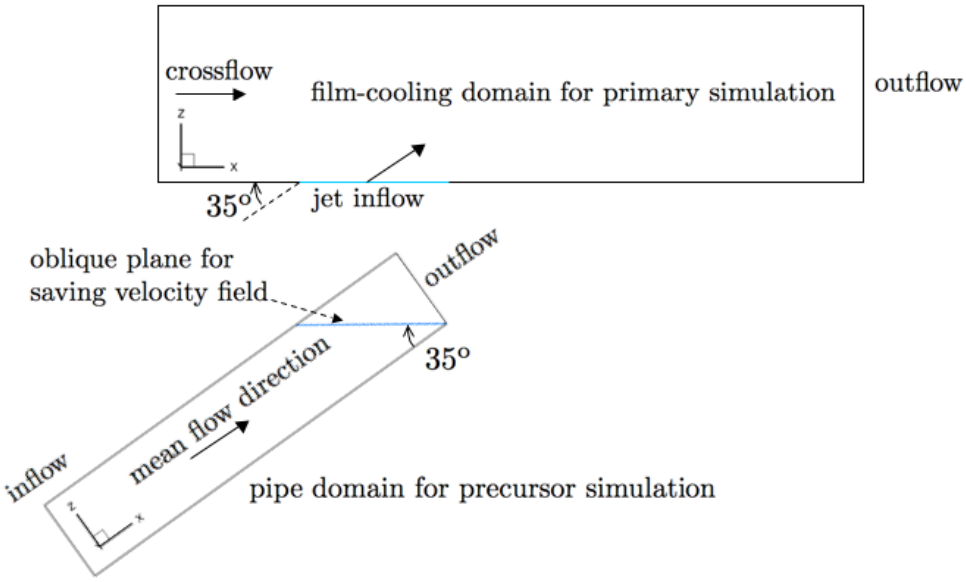


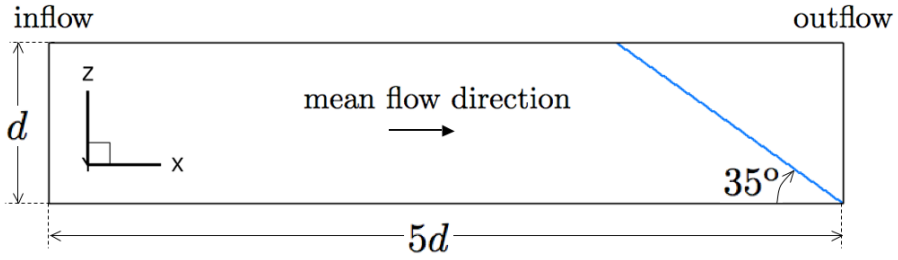
Figure 5.2: Comparison of domains for precursor and film-cooling simulations, where the jet inflow plane and oblique plane through the pipe are parallel.

The following procedure describes how the jet inflow boundary condition was constructed.

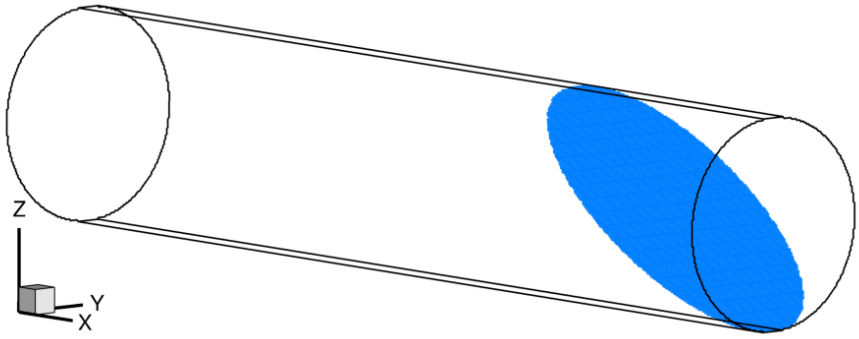
1. *Run precursor LES of turbulent pipe flow.* The computational domain of the pipe is shown in Figure 5.3, where the pipe diameter was d and the domain length of the pipe was $5d$. The characteristic length was the pipe diameter d and the characteristic velocity was the mean friction velocity u_τ. The friction Reynolds number was set as $Re_\tau = u_\tau d/\nu = 764$, which corresponds to a bulk Reynolds number of $Re_b = u_b d/\nu \approx 12,000$. This bulk Reynolds number for the pipe was required owing to the Reynolds number of the film-cooling simulation (Re_∞) and blowing ratio (u_j/u_∞):

$$Re_b = \frac{u_b d}{\nu} = \frac{u_j d}{\nu} = \frac{u_j}{u_\infty}\frac{u_\infty d}{\nu} = \frac{u_j}{u_\infty} Re_\infty = (1.5)(8,000) = 12,000 \tag{5.10}$$

where we have used the fact that the bulk jet velocity in the film-cooling simulation is the same as the bulk pipe velocity from the precursor simulation ($u_j = u_b$).



(a) side view



(b) three-dimensional view

Figure 5.3: Computational domain for precursor LES of turbulent pipe flow. The instantaneous velocity field is extracted along the oblique plane indicated in blue.

The mesh had 128 cells along the diameter and 512 cells in the streamwise direction. No-slip boundary conditions were applied at the walls and a periodic boundary condition was used in the streamwise direction. The flow was driven by a constant mean pressure gradient in the streamwise direction, which is added as a source term to the momentum equation: $\mathbf{F} = -\frac{d\bar{p}^*}{dx^*}\mathbf{i} = 4\mathbf{i}$. A Smagorinsky SGS model [95] was used for the eddy viscosity, with a constant of $C_s = 0.1$. A time-step of $\Delta t = 0.0001(d/u_\tau)$ was used, and the simulation was integrated for $t = 30(d/u_\tau)$ time units.

2. *Select cells along oblique plane passing through pipe domain.*

3. *Save instantaneous velocity data for prescribed number of time-steps.* Data were stored for $0.15(d/u_\tau)$ time units (1500 time-steps) beyond the integration time from step 1 above. Storing more time-steps of velocity data would be preferable to improve the statistics of the jet inflow, but due to storage limitations of the GPU, it was decided to not store more precursor data.

4. *Transform velocity data from coordinate system of precursor domain to coordinate system of film-cooling domain.* The need for this transformation is made clear in Figure 5.4, which shows the orientation of a mesh cell in the precursor simulation and film-cooling simulation. The axes are oriented in the same fashion as in Figure 5.2. Since the y-axis is the same for both simulations, the velocity transformation is a rotation about the y-axis:

$$u = (\cos\theta)u_{precursor} + (\sin\theta)w_{precursor} \tag{5.11}$$

$$v = v_{precursor} \tag{5.12}$$

$$w = -(\sin\theta)u_{precursor} + (\cos\theta)w_{precursor} \tag{5.13}$$

   where θ is the rotation angle in the counter-clockwise direction ($\theta = 2\pi - \alpha$).

5. *Save precursor library to file.*

6. *Load precursor library from file into film-cooling simulation.*

7. *Interpolate precursor data onto jet inflow area.* The area saved from the precursor was a 128 x 128 cell plane of data, where only the velocities in the elliptical intersection of the pipe are used. The data were interpolated onto the jet inflow area, which was an elliptical area with 128 cells in the streamwise direction (major axis) x 64 cells in the

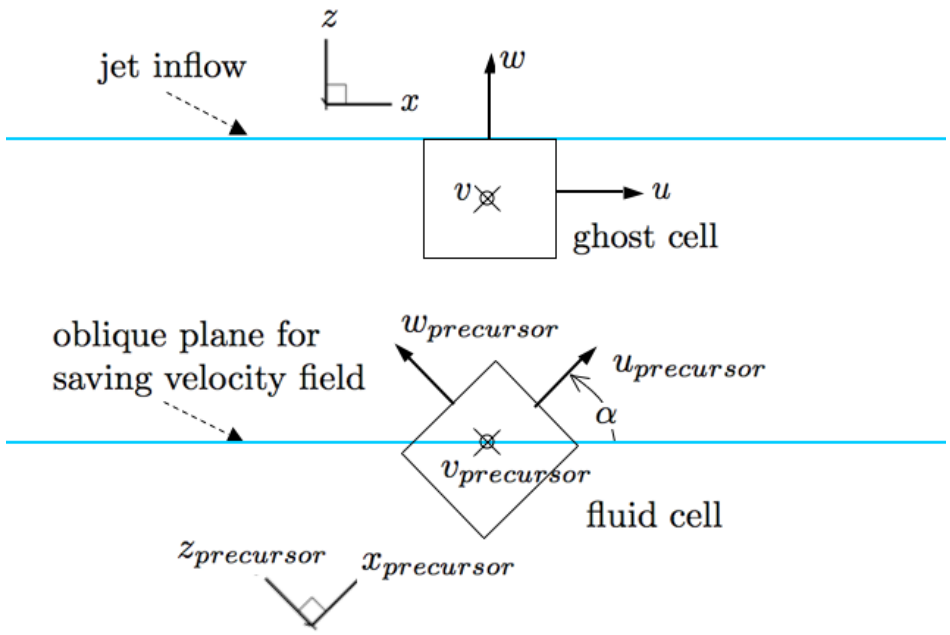


Figure 5.4: Cell orientation for precursor and film-cooling simulations.

spanwise direction (minor axis) for the fine mesh, and 64 cells (major axis) x 64 cells (minor axis) for the coarse mesh.

8. *Scale the velocity components from the precursor such that they are non-dimensionalized by the characteristic velocity used in the film-cooling simulation.* The scaling was done as follows:

$$\frac{\phi}{u_\infty} = \frac{u_j}{u_\infty}\frac{(\phi/u_\tau)}{(u_b/u_\tau)} \tag{5.14}$$

where ϕ is a velocity component (u, v, w) from the precursor simulation, u_b is the bulk velocity in the pipe from the precursor simulation, and u_j/u_∞ is the blowing ratio. Note that the bulk jet velocity is equal to the bulk pipe velocity $u_j = u_b$.

9. *Apply precursor data as jet inflow boundary condition.* For each time-step of the film-cooling simulation, the velocity field at the jet inflow is changed to the next time-step of the precursor data. This can be done because the physical time-step for the film-cooling simulation is approximately the same as the physical time-step for the precursor

simulation, and thus the temporal evolution is the same; this is proven mathematically in Appendix B. After all time-steps of the precursor data have been used, the data are recycled by starting with the first time-step in the data set.

5.4.7 Results from Precursor LES of Turbulent Pipe Flow

A "cut-away" view of the instantaneous streamwise velocity from the precursor LES of turbulent pipe flow is shown in Figure 5.5. The mean flow is directed in the positive x-direction. This figure provides a qualitative description of the turbulent flow, where the contour distributions indicate strong mixing from the turbulent eddies.

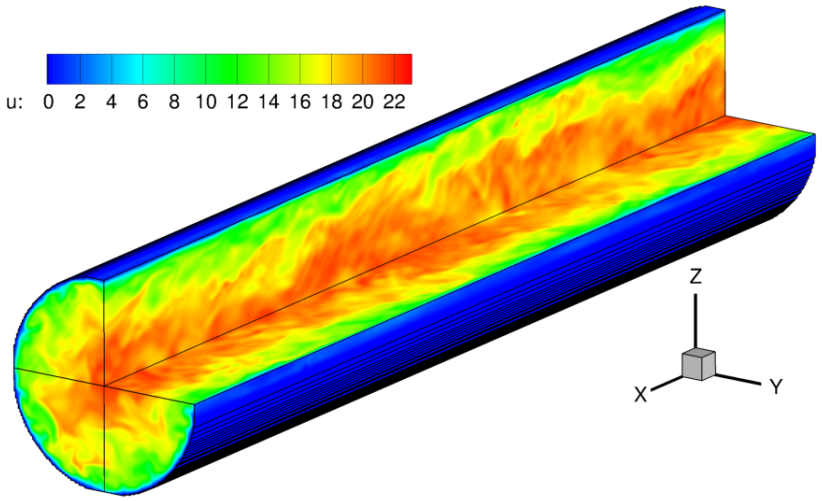
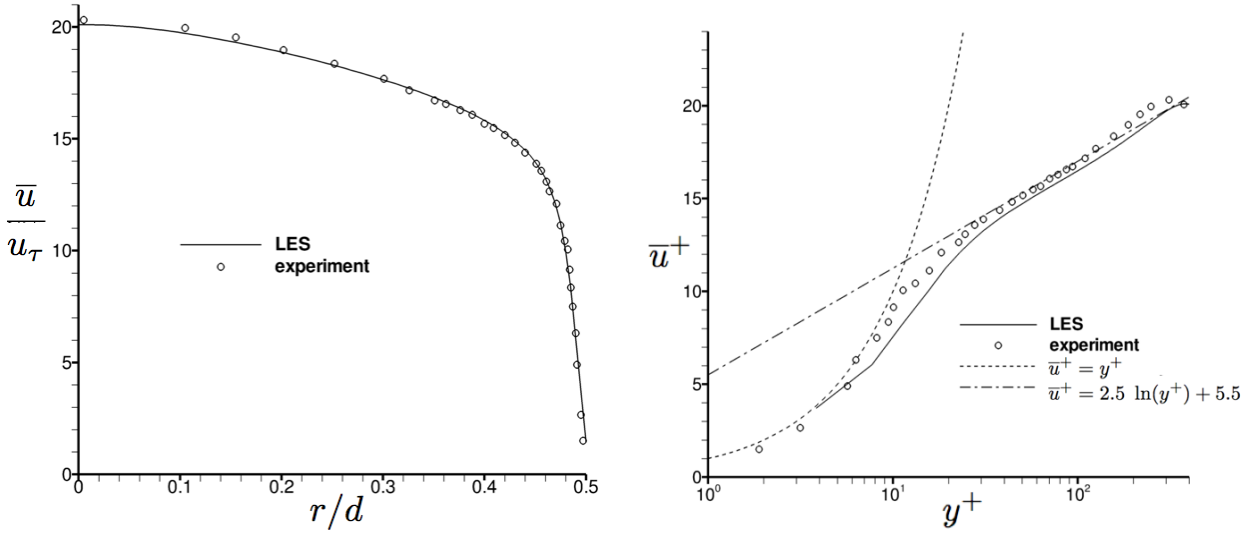


Figure 5.5: Cut-away view of the instantaneous streamwise velocity for the turbulent pipe precursor LES.

The mean statistics were collected for the last $20(d/u_\tau)$ time units of the simulation and the root-mean-square statistics were collected for the last $10(d/u_\tau)$ time units of the simulation. The mean streamwise velocity profile along the radius is shown in Figure 5.6(a) and compared with experimental LDV measurements of den Toonder and Nieuwstadt [110], where their experimental values were taken for $Re_b = 10,000$. In addition, the same mean profile is plotted versus wall units in Figure 5.6(b) and compared to experimental data and

the law of the wall. Excellent agreement with experiment is observed in (a) and reasonable agreement with experiment and the law of the wall is observed in (b). Root-mean-square statistics are shown in Figure 5.7, and a comparison with experiment shows good agreement away from the wall, but an under-prediction close to the wall.



(a) (b)

Figure 5.6: Mean streamwise velocity profile from precursor LES of turbulent pipe flow (at $Re_b \approx 12,000$) compared with experimental LDV measurements of den Toonder and Nieuwstadt [110] (at $Re_b = 10,000$).

5.5 Results for LES of Film-Cooling Flow

Results will now be presented and discussed for the film-cooling flow. The baseline and micro-ramp cases were simulated using coarse and fine meshes, but only the fine mesh results will be shown. The results from the coarse mesh will only be used to assess the effect of mesh resolution on the film-cooling effectiveness.

5.5.1 Instantaneous Field

To visualize the three-dimensional structure of the jet, an isosurface of instantaneous temperature for the baseline case is shown in Figure 5.8 at $T^* = 0.5$, which is the midpoint of

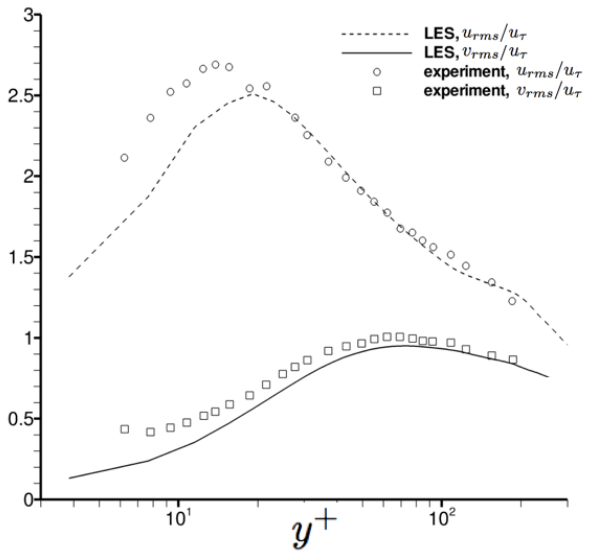
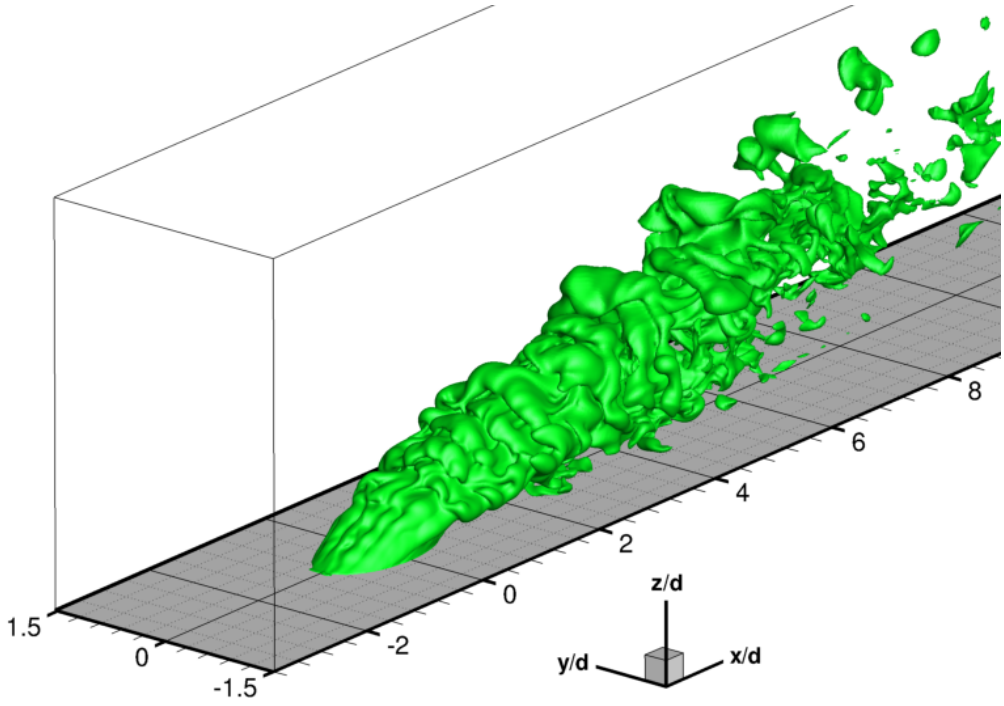


Figure 5.7: Root-mean-square statistics from precursor LES of turbulent pipe flow (at $Re_b \approx 12,000$) compared with experimental LDV measurements of den Toonder and Nieuwstadt [110] (at $Re_b = 10,000$).

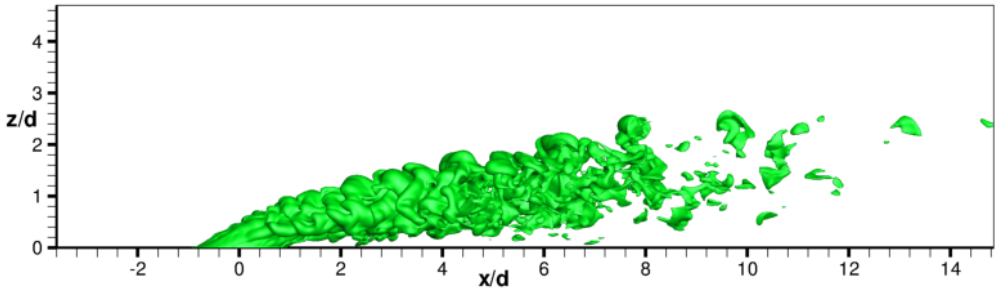
the two extrema of the temperature field ($T^*_{\max} = 1, T^*_{\min} = 0$). Lateral spreading of the jet is observed in the $x-y$ plane (top view). In the $x-z$ plane (side view), vertical penetration of the jet into the cross-flow (jet lift-off) is observed to increase in the streamwise direction.

Figure 5.9 shows the instantaneous velocity magnitude of the coolant flow at jet inflow ($z/d = 0$) for different solution times, which demonstrates the unsteadiness of the inflow. As noted previously, this flowfield was created from a precursor simulation and was stored as a library that is used to update the inflow for each time-step of the simulation. Figure 5.10 presents a comparison of the instantaneous velocity magnitude at midspan ($y/d = 0$) for the baseline and micro-ramp cases. As expected, the velocity magnitude is highest in the jet near-field and dissipates in the far-field. We observe that the instantaneous jet trajectory is not altered by the micro-ramp. On the leeward side of the micro-ramp there is a low speed region where flow is locally separated.

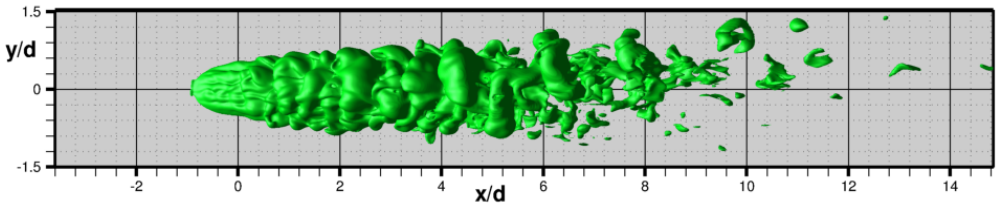
Figure 5.11 shows a comparison between the instantaneous temperature fields of the baseline flow (left column) and flow with a micro-ramp (right column). These fields are plotted along the midspan of the domain ($y/d = 0$) at four different non-dimensional times during



(a) three-dimensional view



(b) side view



(c) top view

Figure 5.8: Visualization of jet in baseline flow using isosurface of instantaneous temperature at $T^* = 0.5$.

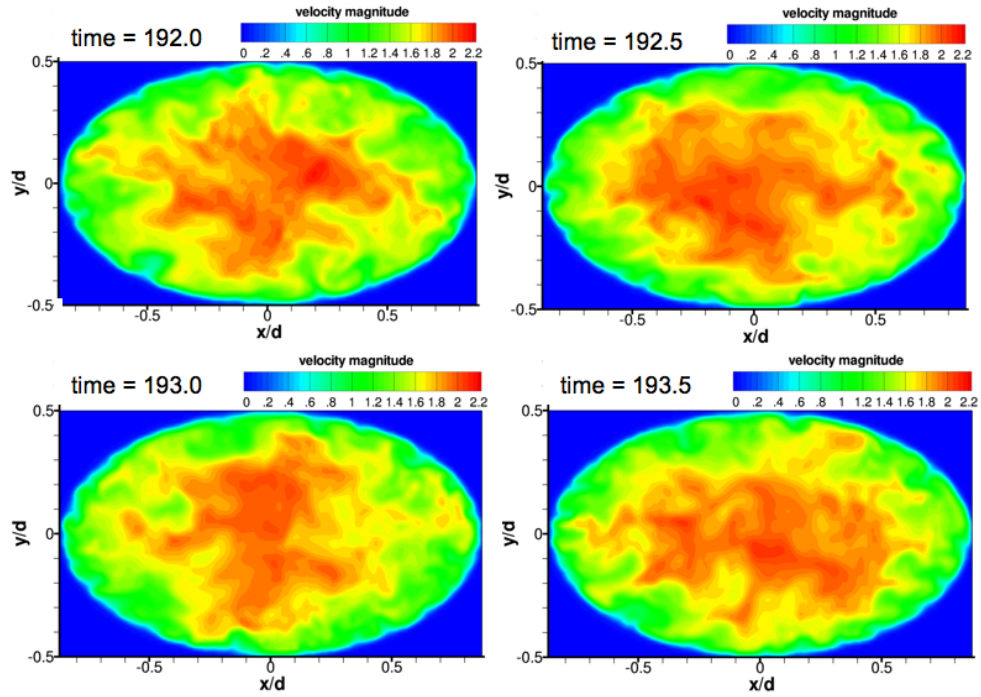


Figure 5.9: Instantaneous velocity magnitude of flow at jet inflow (z/d=0) for different solution times.

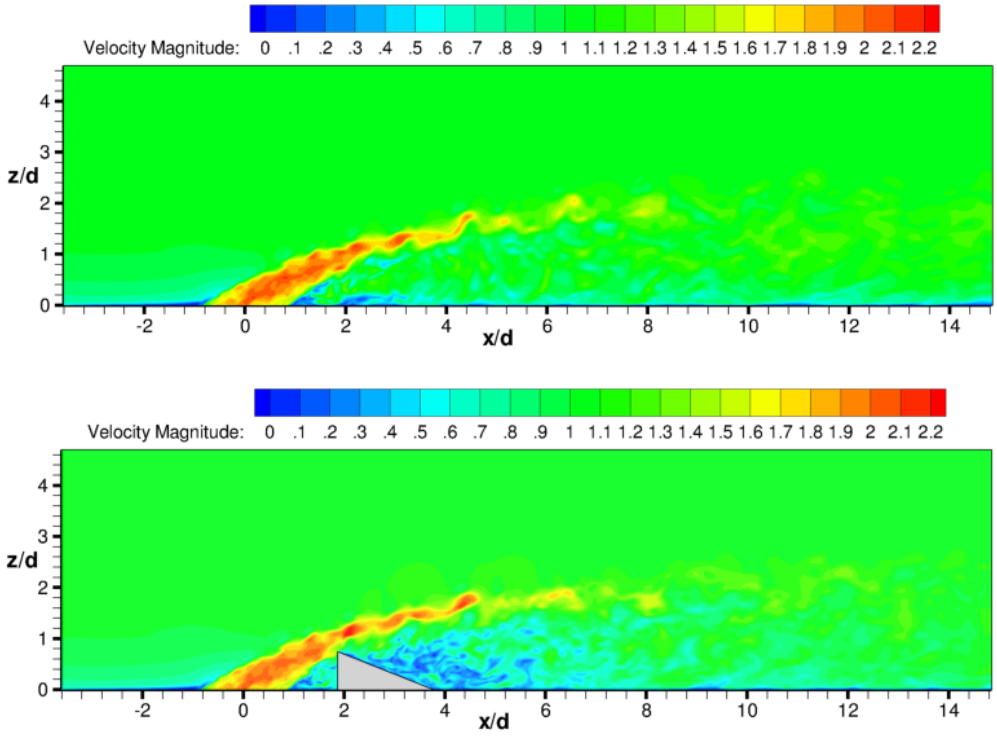


Figure 5.10: Instantaneous velocity magnitude at midspan ($y/d = 0$).

the simulation. The windward side of the jet has a wavy appearance due to the presence of shear-layer vortices. These vortices convect downstream and grow into larger structures in the jet far-field. This behavior is seen with or without the micro-ramp. Downstream of the jet for the baseline flow, the coolant flow is mostly separated from the surface. With the micro-ramp placed downstream of the jet, more violent mixing is observed in the jet near-field and more regions of lower temperature near the wall are observed in the jet far-field. This latter observation indicates that there is better transport of coolant toward the wall when using a micro-ramp, effectively maintaining beneficial jet attachment to increase cooling effectiveness.

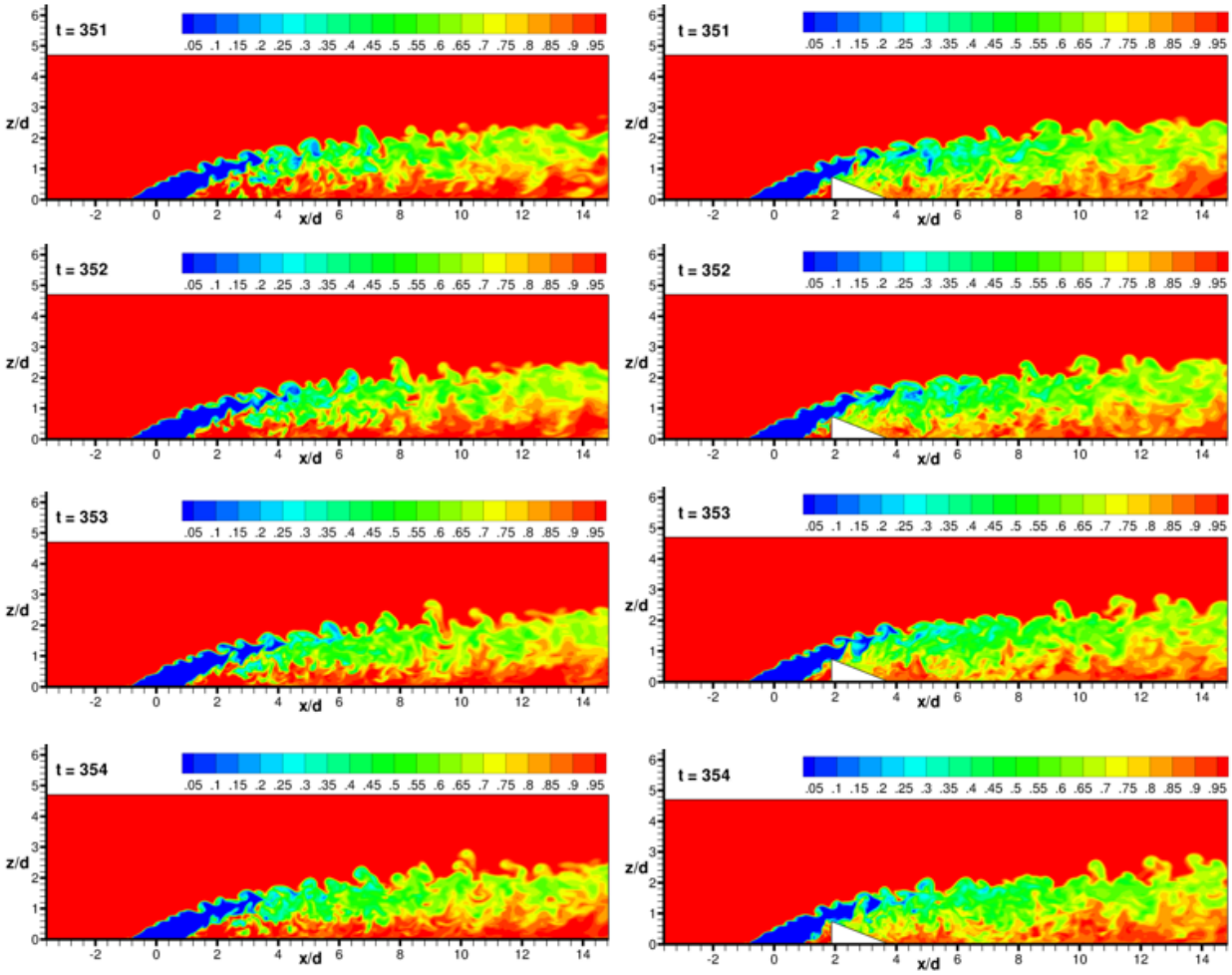


Figure 5.11: Comparison of instantaneous temperature at midspan ($y/d = 0$) for the baseline case (left column) versus the micro-ramp case (right column) at different solution times.

5.5.2 Mean Field

Contours of time-averaged velocity magnitude for the jet inflow are shown in Figure 5.12. The jet velocity distribution is somewhat asymmetric. The lack of symmetry may be due to the jet inflow data being recycled every $1.5(d/u_\infty)$ time units, resulting in a smaller sample size.

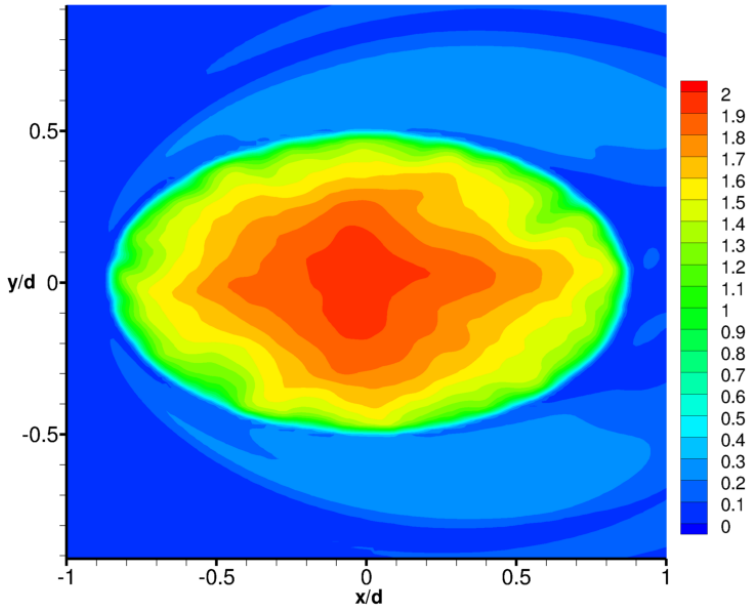


Figure 5.12: Mean velocity magnitude at jet inflow ($z/d = 0$).

The mean velocity magnitude at midspan is shown in Figure 5.13. It can be seen that the shear-layer vortices along the windward side of the jet are averaged out in the mean flow. This observation emphasizes the importance of using simulation methods that can capture the unsteady behavior of the flow. However, the time-averaged velocity resembles the behavior of the instantaneous flow, where a region of low velocity is created in the wake of the micro-ramp due to the separated flow. The vertical penetration of the jet is relatively unaffected by the micro-ramp.

To examine the jet trajectory and jet lateral spreading, streamlines of the mean flow are plotted in Figure 5.14, where a rake of 15 equally-space streamlines was placed along the jet inflow major axis (side view) and jet inflow minor axis (top view). We note little difference in

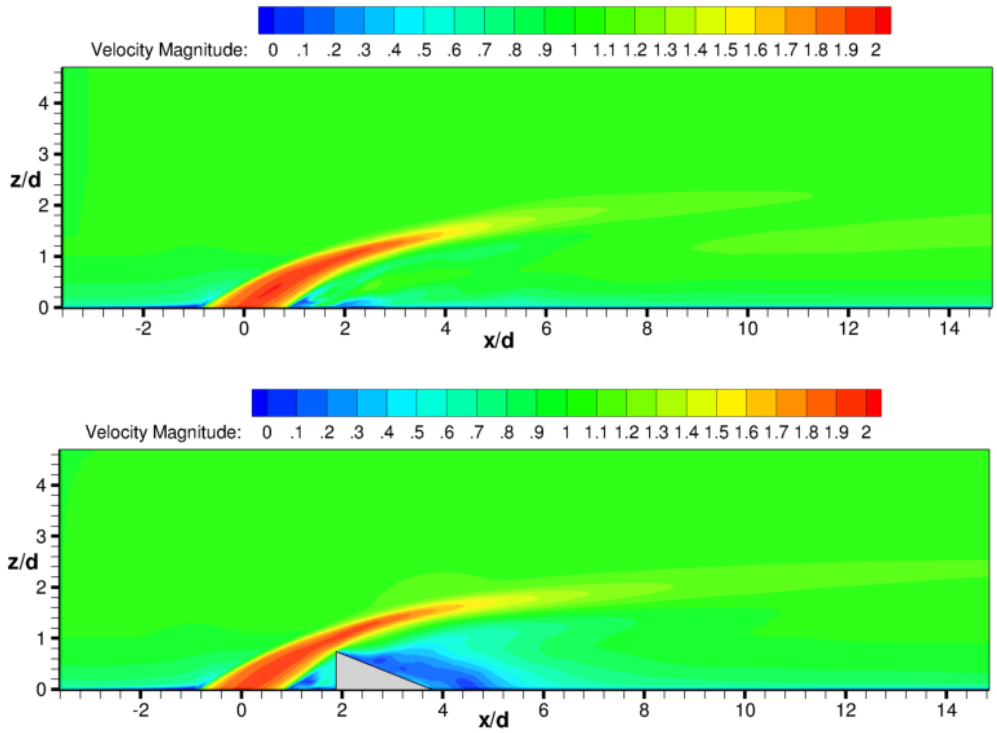
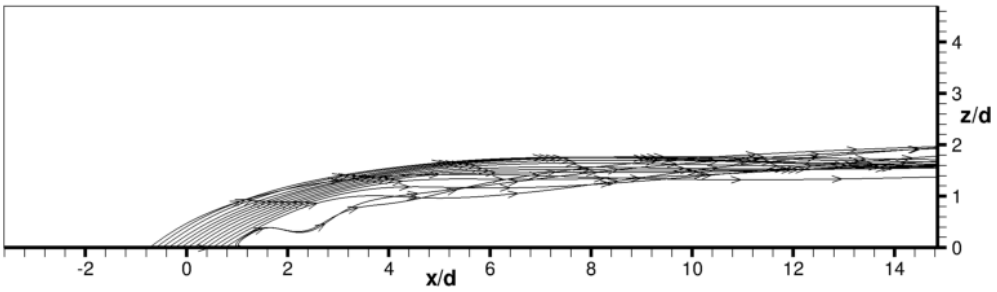


Figure 5.13: Mean velocity magnitude at midspan ($y/d = 0$).

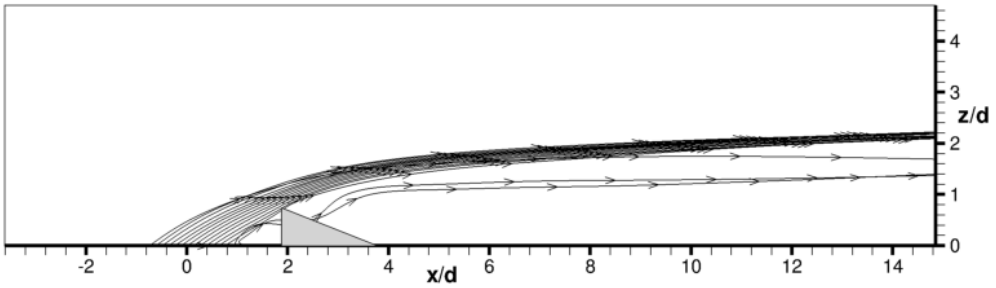
jet trajectory and lateral spreading between the baseline and micro-ramp flows, except that the micro-ramp appears to compress the streamlines such that they converge downstream and that the jet trajectory is a little higher near the outflow for the micro-ramp case. The primary reason for similar jet trajectories is that the jet flow moves over the micro-ramp and there is apparently not enough counter-vorticity generated by the micro-ramp to prevent jet lift-off. Placement of the micro-ramp closer to the jet may have a larger effect.

To examine the streamwise evolution of the flow field, eight streamwise planes were selected as survey locations, as indicated in Figure 5.15. The mean streamwise vorticity and mean temperature will be plotted along these planes. Planes are clustered near the ramp since the flow changes rapidly in that region. First, we will examine the vorticity field along these planes. The non-dimensional mean streamwise vorticity ($\overline{\omega}_x^* = \overline{\omega}_x d/u_\infty$) is

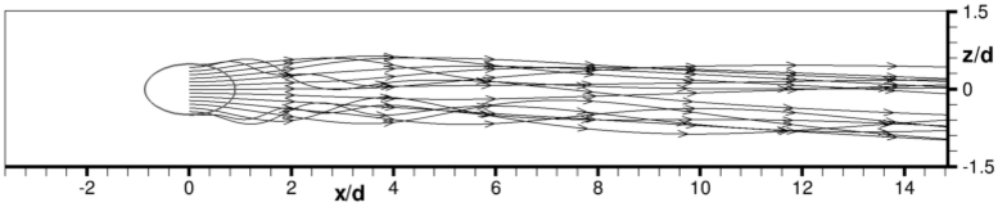
$$\overline{\omega}_x^* = \frac{\partial \overline{w}^*}{\partial y^*} - \frac{\partial \overline{v}^*}{\partial z^*} \tag{5.15}$$



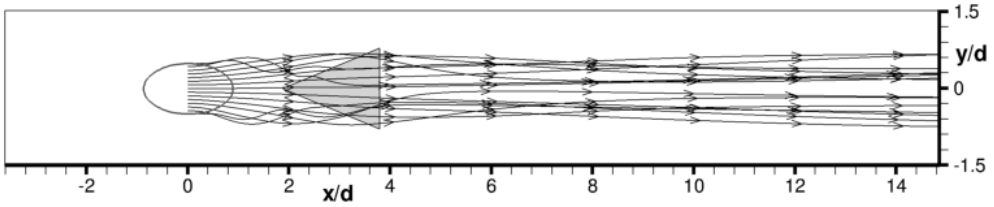
(a) side view, baseline flow



(b) side view, flow with micro-ramp



(c) top view, baseline flow



(d) top view, flow with micro-ramp

Figure 5.14: Mean streamlines from jet, where a rake of 15 equally-spaced streamlines was placed along the jet inflow major axis (for side view) and minor axis (for top view).

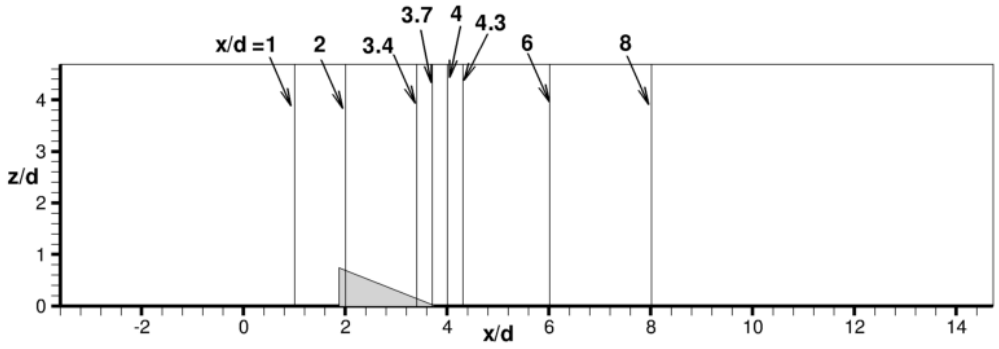
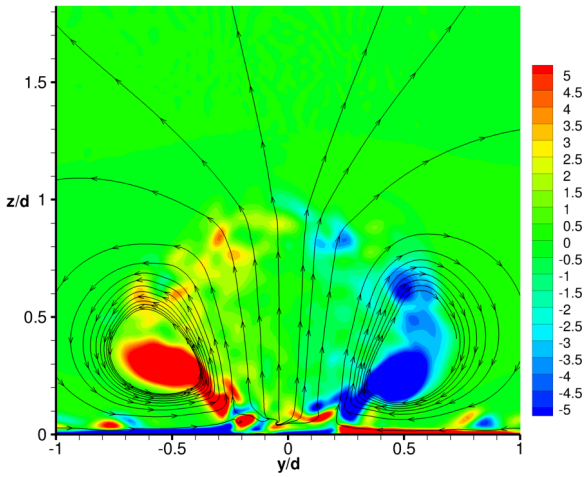


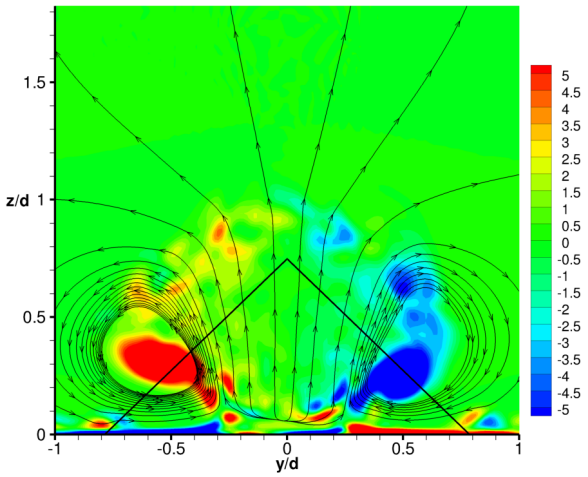
Figure 5.15: Location of streamwise survey planes.

and is plotted in Figure 5.16 along the eight streamwise planes. Mean streamlines contained within the plane are superimposed on the vorticity to help visualize the flowfield. Also note that the cross-flow is directed in the positive x-direction (out of the paper). The left column is for the baseline flow, and the right column is for the flow with a micro-ramp. For the micro-ramp case, the triangular projection of the entire micro-ramp cross-section is indicated with thick black lines, and for planes that intersect the micro-ramp ($x/d = 2,\ 3.4,\ 3.7$) the cross-section of the micro-ramp is shown in gray.

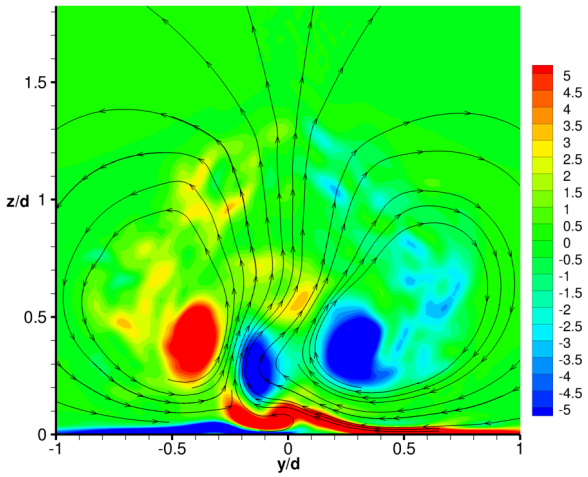
At $x/d = 1$ the vorticity and streamlines clearly show a symmetric counter-rotating vortex pair (CRVP) in the jet, where the mean flow is directed such that an upwash component exists between the two vortices. We see that the flow fields for the baseline and micro-ramp cases are similar at $x/d = 1$, since this location is upstream of the micro-ramp. At $x/d = 2$ the plane is located just downstream of the leading-edge of the micro-ramp; here we see vorticity generated at the micro-ramp surface that is opposite in sign to the CRVP vorticity. At $x/d = 3.4$ the counter-rotating vortices generated by the micro-ramp are visible, and a downwash direction in the mean flow is present between these vortices. Compared to the baseline flow, the micro-ramp vortices have weakened the jet vortices through vorticity cancellation, as evidenced by the smaller regions of vorticity in the CRVP. A little farther downstream at $x/d = 3.7$ a similar vorticity cancellation behavior is observed, and we also see that the vorticity levels have decreased in both the jet CRVP and micro-ramp vortices



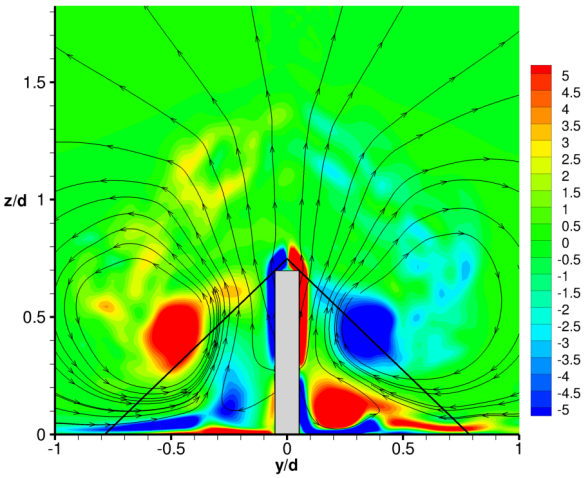
(a) $x/d = 1$, no micro-ramp



(b) $x/d = 1$, with micro-ramp

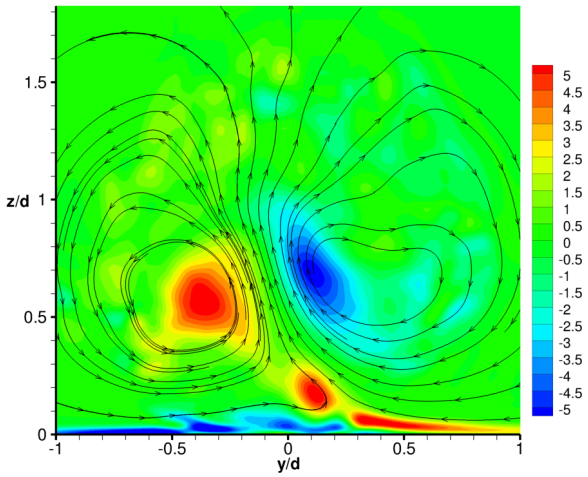


(c) $x/d = 2$, no micro-ramp

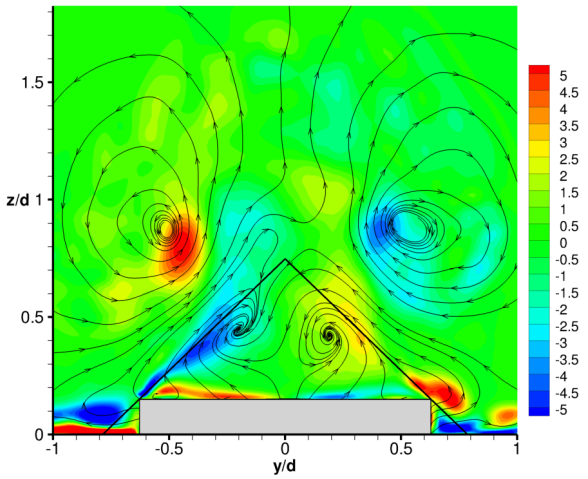


(d) $x/d = 2$, with micro-ramp

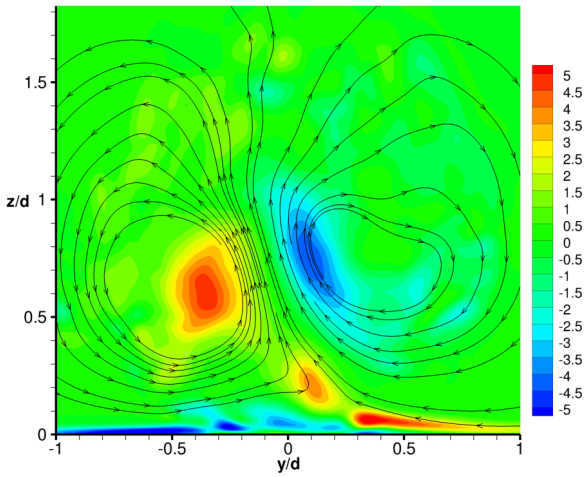
Caption on page 125.



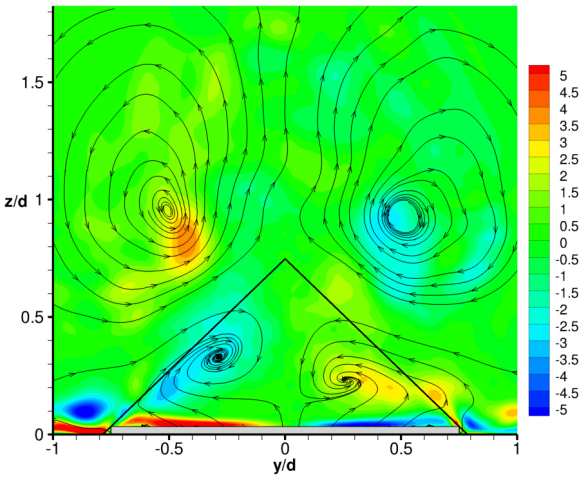
(e) $x/d = 3.4$, no micro-ramp



(f) $x/d = 3.4$, with micro-ramp

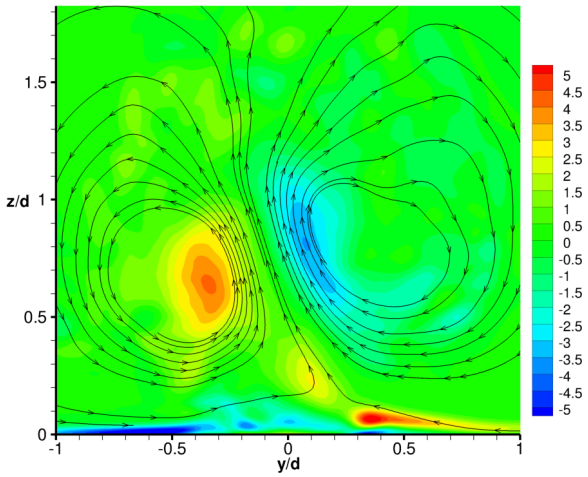


(g) $x/d = 3.7$, no micro-ramp

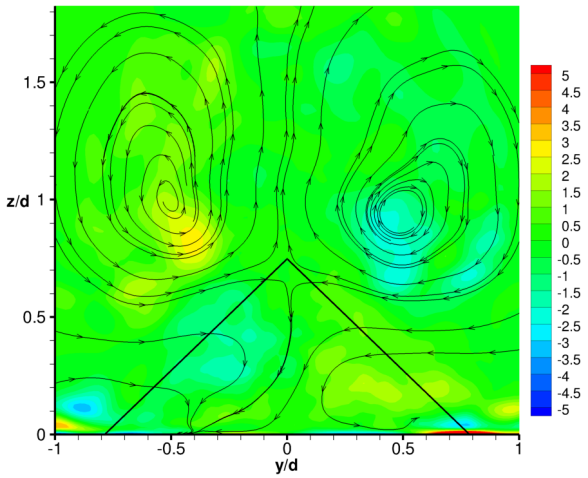


(h) $x/d = 3.7$, with micro-ramp

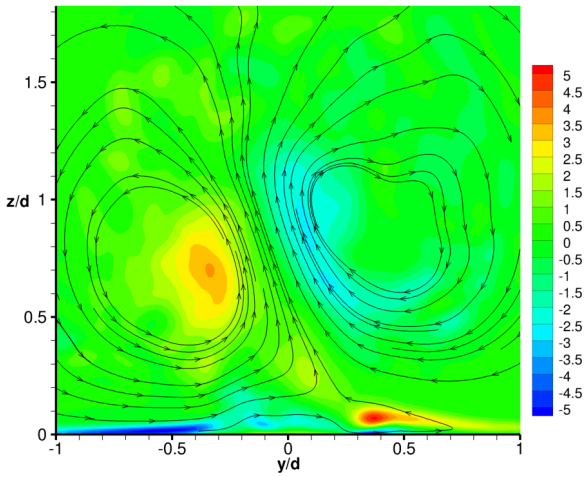
Caption on page 125.



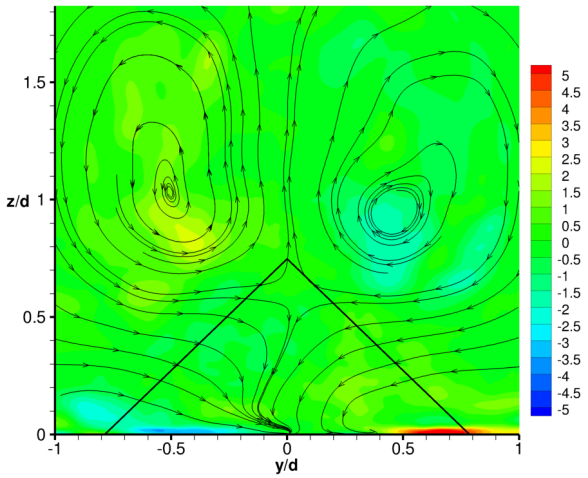
(i) $x/d = 4$, no micro-ramp



(j) $x/d = 4$, with micro-ramp

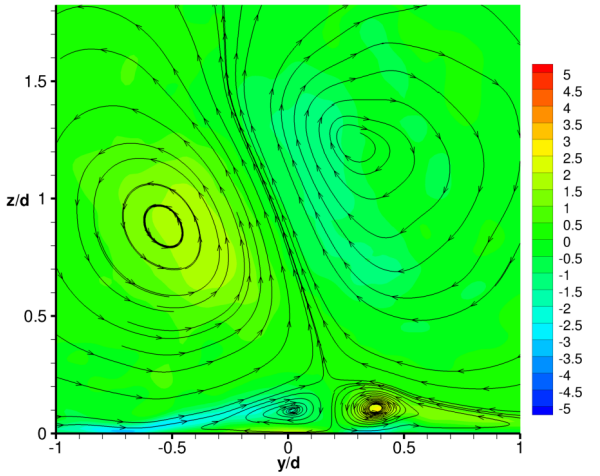


(k) $x/d = 4.3$, no micro-ramp

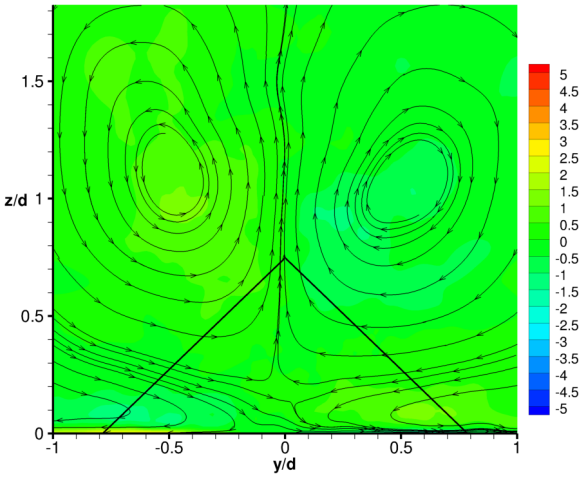


(l) $x/d = 4.3$, with micro-ramp

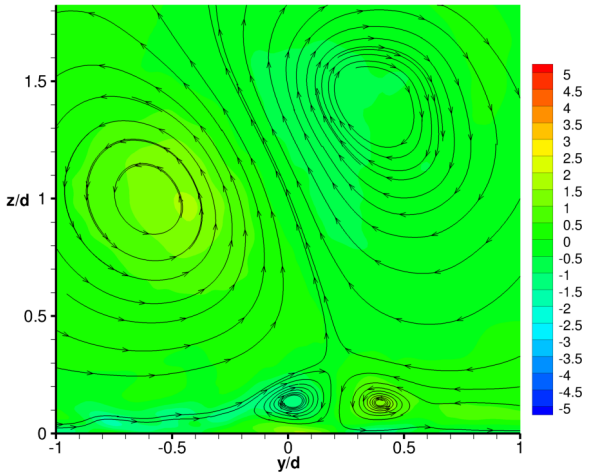
Caption on page 125.



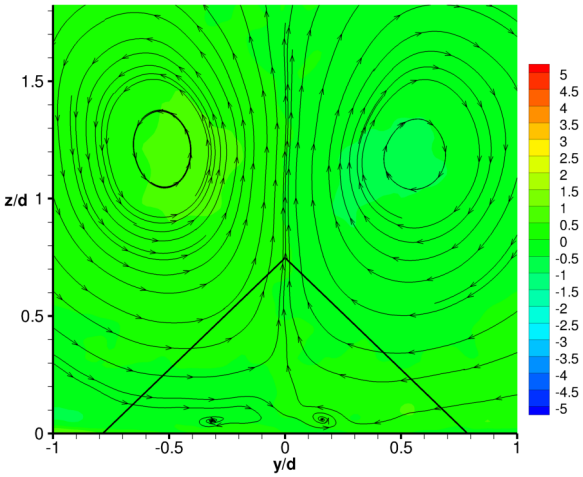
(m) $x/d = 6$, no micro-ramp



(n) $x/d = 6$, with micro-ramp



(o) $x/d = 8$, no micro-ramp



(p) $x/d = 8$, with micro-ramp

Figure 5.16: Contours of mean streamwise vorticity overlaid with streamlines in the plane.

compared with the vorticity field at $x/d = 3.4$. At $x/d = 4$ the vortices from the micro-ramp have decayed, indicating they were very weak. However, the beneficial downwash effect induced by these vortices persists downstream until $x/d = 8$, where the micro-ramp vortices are hardly visible. At $x/d = 6$ and $x/d = 8$ for the baseline case, the jet appears to be driving a secondary vortex pair near the wall that is opposite in sense to the CRVP of the jet. These vortices have a negligible impact on temperature distribution, which will become evident by examining plots of film-cooling effectiveness (shown later). Thus the jet CRVP remains the dominant structure affecting film-cooling at these locations. Lastly, it can be seen in Figure 5.16 that the jet CRVP in the flow with no micro-ramp is not symmetric about the midspan of the domain ($y/d = 0$), which is probably the result of an insufficient amount of time-averaging.

We now turn our attention to the mean temperature field. The mean temperature at midspan is shown in Figure 5.17. We again see that shear-layer vortices (as well as other downstream eddy structures) have been removed via time-averaging, leaving a temperature field with relatively smooth variations. The most important observation from these plots is that for the micro-ramp case the mean temperature is lower near the wall, indicating better cooling.

Figure 5.18 shows mean temperature along the wall of the domain at $z/d = 0$. At the leading-edge of the jet, the low temperature region is caused by a horseshoe vortex that entrains coolant from the jet and redistributes it along the wall. The horseshoe vortex clearly has a beneficial effect on film-cooling performance; the dynamics of the horseshoe vortex will be examined further in section 5.5.3. Now looking downstream of the jet in Figure 5.18, we see that an envelope of reduced temperature exists along the centerline for both cases, but the flow with a micro-ramp has a wider envelope and a lower wall temperature at the centerline. Outside of this envelope, the wall experiences almost no cooling, and it remains at the temperature of the cross-flow.

The streamwise evolution of the mean temperature is shown in Figure 5.19 for the stream-

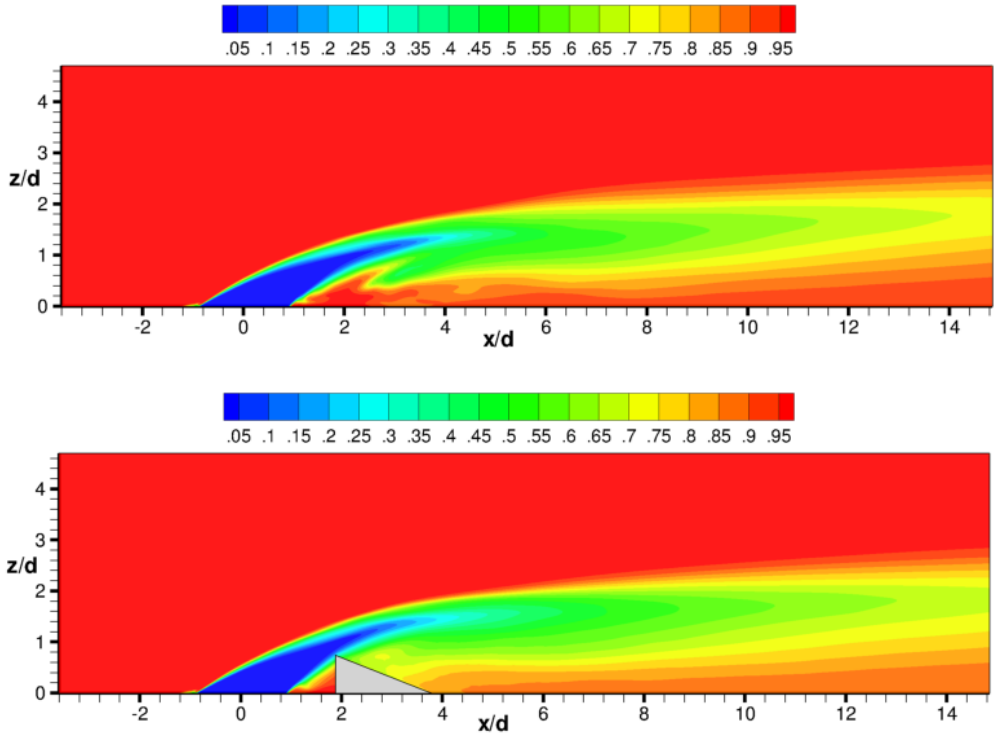


Figure 5.17: Mean temperature at midspan ($y/d = 0$).

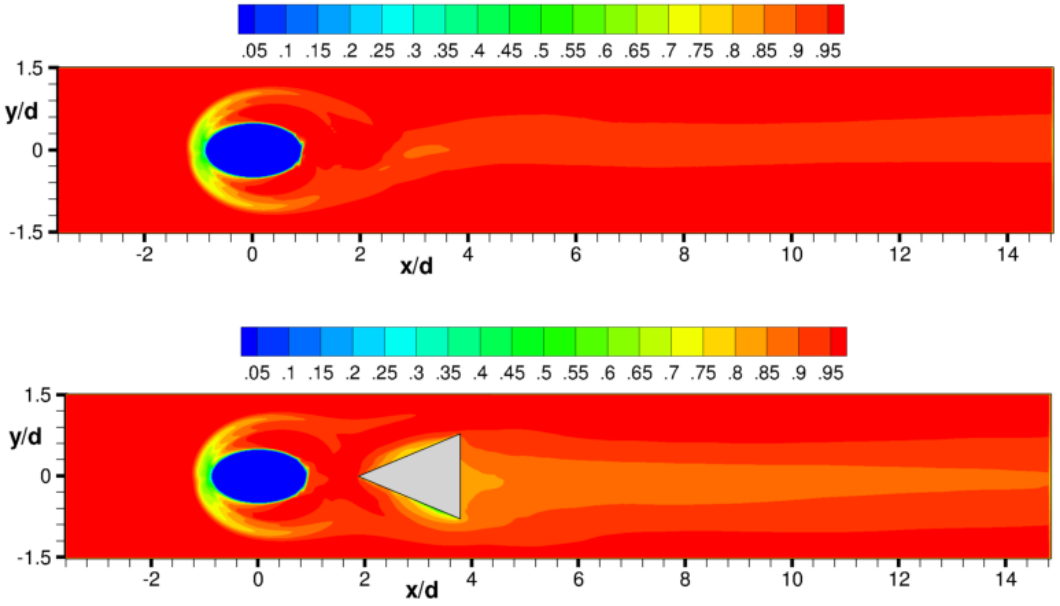


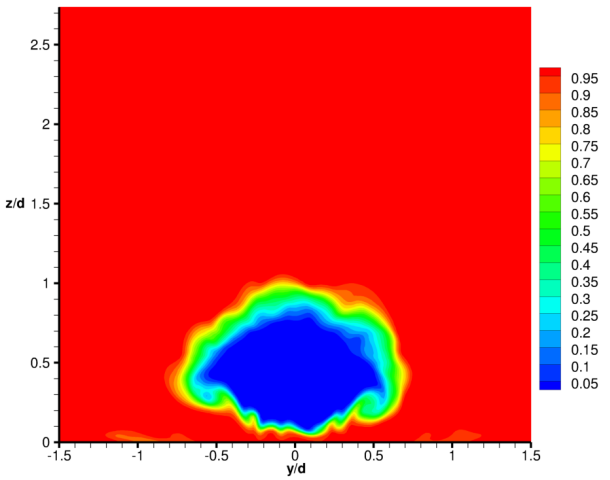
Figure 5.18: Mean temperature along wall of domain ($z/d = 0$).

wise planes indicated in Figure 5.15. This figure shows how coolant is distributed vertically and laterally from the jet for both the baseline flow and flow with the micro-ramp. The cross-flow is directed in the positive x-direction (out of the paper). The results are presented in the same format as was done previously for the vorticity, where the left column is for the baseline flow, the right column is for the flow with a micro-ramp, and the micro-ramp's full triangular projection is indicated with thick black lines and its intersection with the plane is indicated in gray. At $x/d = 1$ we see that the baseline and micro-ramp flows are identical, since the micro-ramp does not appear to have an upstream influence. At $x/d = 2$ the jet is relatively unaffected by the micro-ramp since the jet has just passed the leading edge. At $x/d = 3.4$ and 3.7, however, a drastic difference is observed: in the baseline flow the temperature contours are bulging up away from the wall and we see minimal attachment of the jet, but with a micro-ramp the contours are bulging down toward the wall with improved jet attachment. The temperature distributions at the rest of the downstream planes reveal a similar behavior.

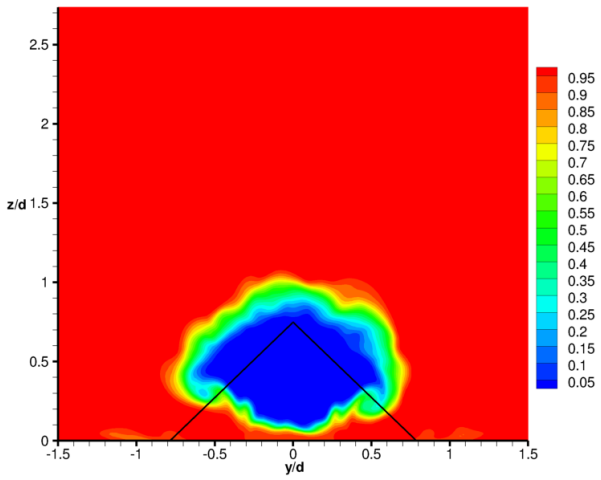
The improved cooling produced by the micro-ramp can be explained by the counter-rotating vortices created by the micro-ramp, as was shown in Figure 5.16. For the baseline flow, the mean velocity vectors indicated that there is primarily an upwash effect between the jet and wall, owing to the effect of the CRVP in the jet. This resulted in transport of coolant away from the wall, as indicated by the upward bulging of the temperature contours. However, for the micro-ramp flow we see primarily a downwash effect between the jet and wall, owing to the effect of the vortices from the micro-ramp that counteract the CRVP in the jet. This resulted in transport of coolant toward the wall as indicated by the downward bulging temperature contours.

5.5.3 Coherent Structures

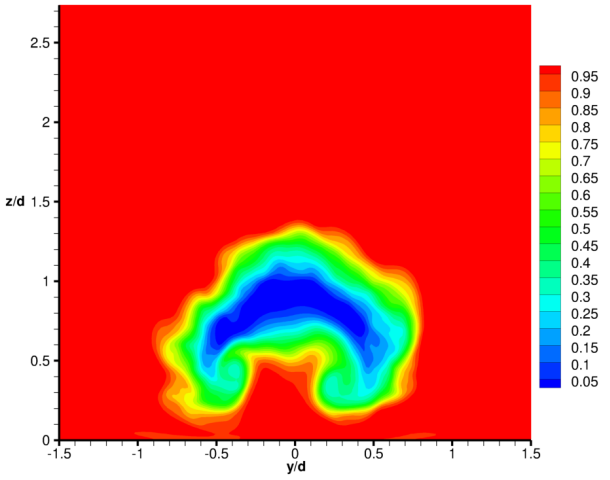
As noted earlier, there are a number of coherent turbulent structures that can appear in a JICF. These include shear-layer vortices at the leading edge of the jet, a counter-rotating



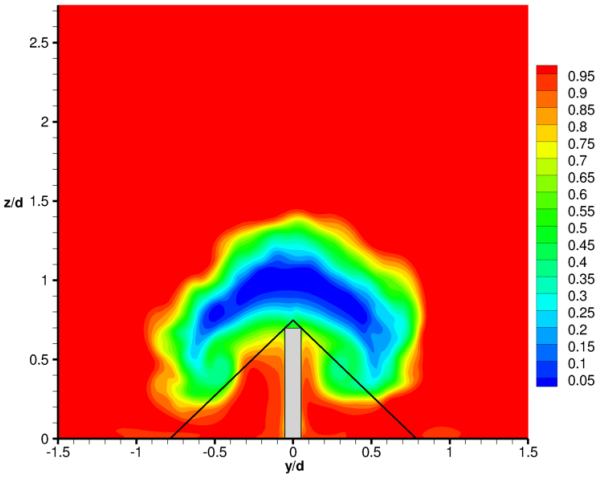
(a) $x/d = 1$, no micro-ramp



(b) $x/d = 1$, with micro-ramp

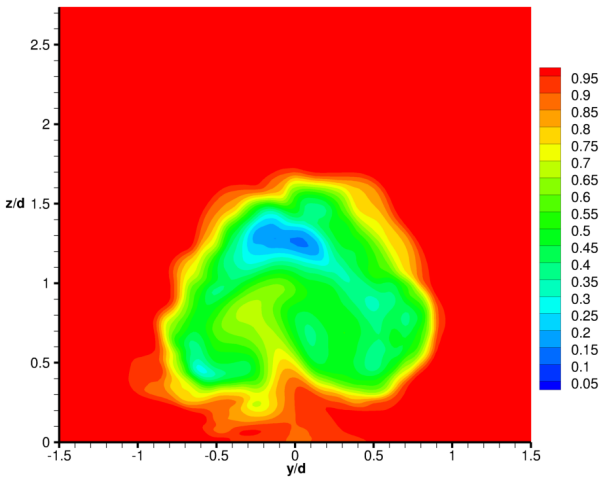


(c) $x/d = 2$, no micro-ramp

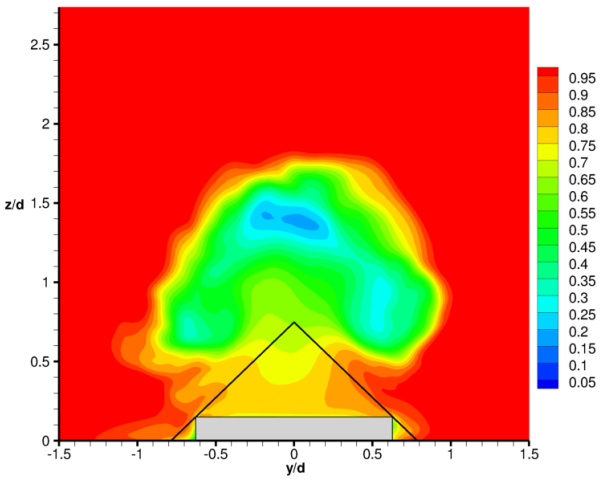


(d) $x/d = 2$, with micro-ramp

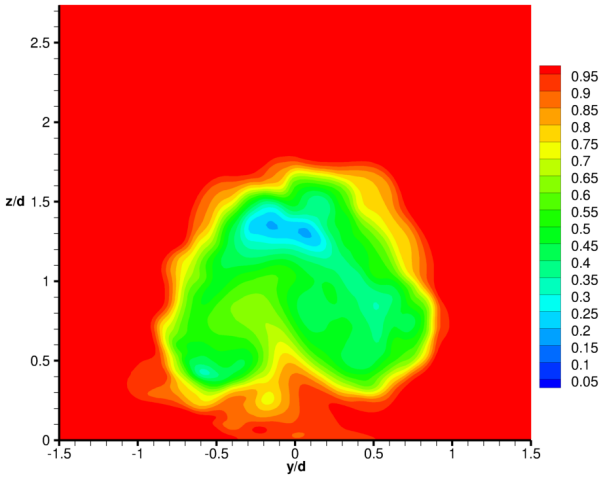
Caption on page 132.



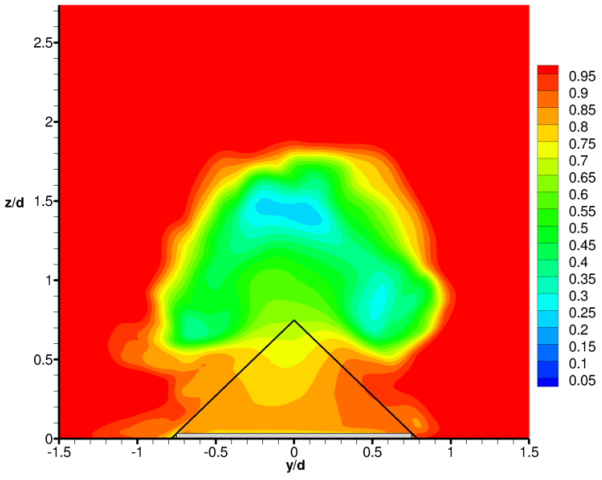
(e) $x/d = 3.4$, no micro-ramp



(f) $x/d = 3.4$, with micro-ramp

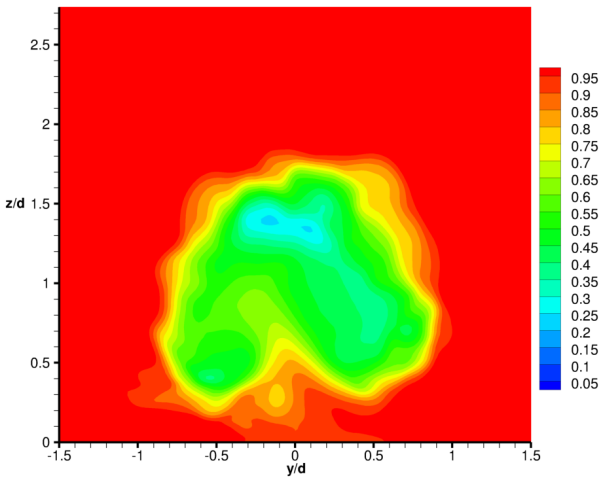


(g) $x/d = 3.7$, no micro-ramp

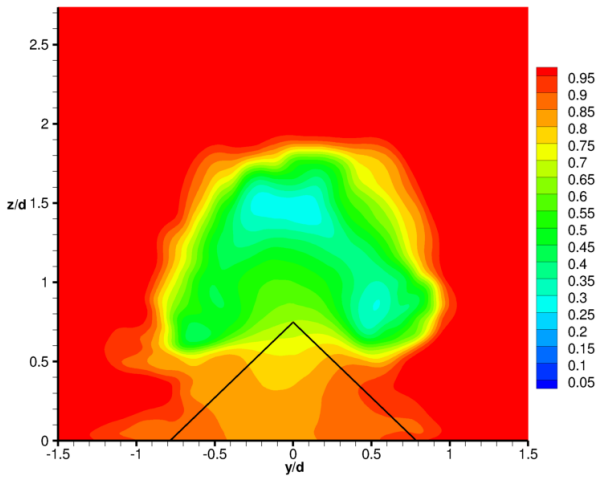


(h) $x/d = 3.7$, with micro-ramp

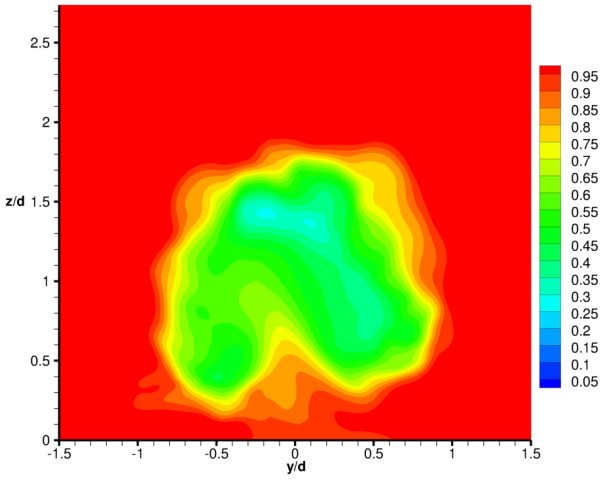
Caption on page 132.



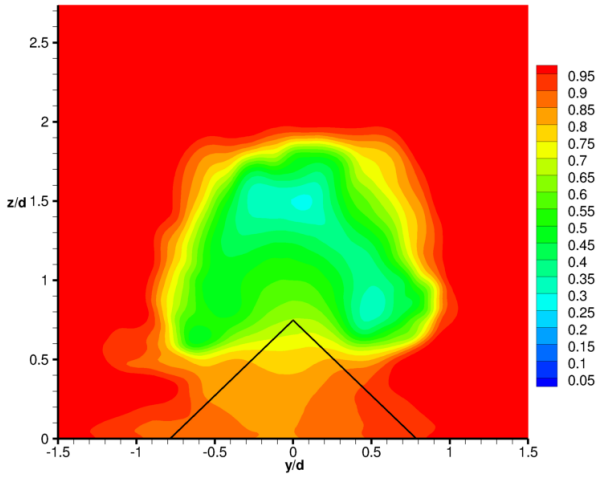
(i) $x/d = 4$, no micro-ramp



(j) $x/d = 4$, with micro-ramp

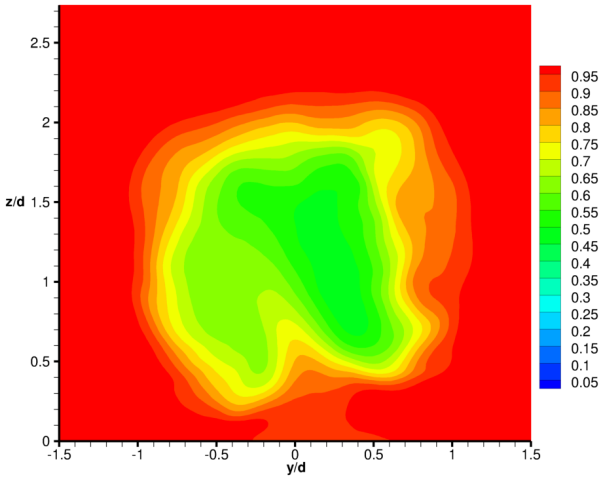


(k) $x/d = 4.3$, no micro-ramp

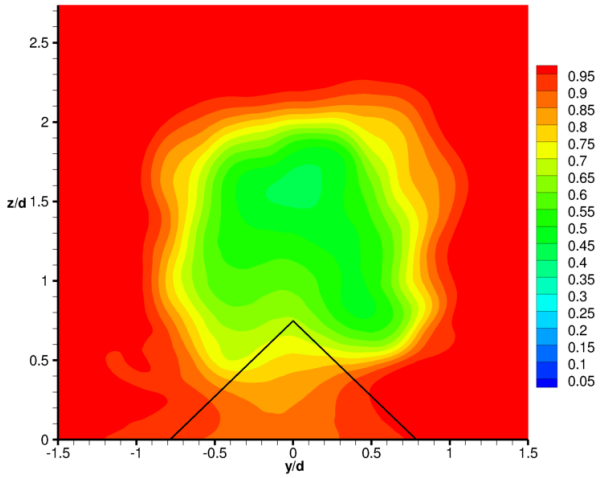


(l) $x/d = 4.3$, with micro-ramp

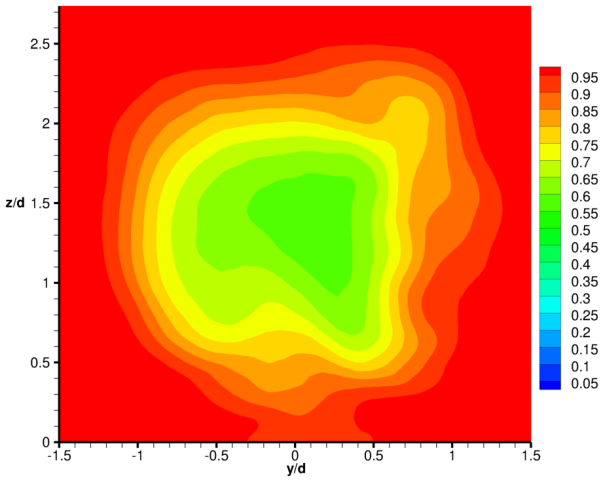
Caption on page 132.



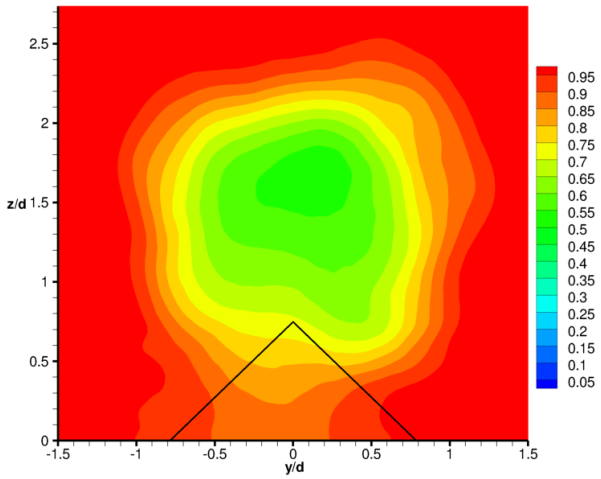
(m) $x/d = 6$, no micro-ramp



(n) $x/d = 6$, with micro-ramp



(o) $x/d = 8$, no micro-ramp



(p) $x/d = 8$, with micro-ramp

Figure 5.19: Mean temperature contours at streamwise planes.

vortex pair (CRVP) in the jet, horseshoe vortices around the base of the jet, and wake vortices downstream of the jet. The shear-layer vortices are not visible in the mean flow, as was indicated earlier (cf. Figure 5.17). Instead, the instantaneous velocity field must be used to visualize shear-layer vortices. This can be done by using the non-dimensional instantaneous spanwise vorticity ($\omega_y^* = \omega_y d/u_\infty$), computed as

$$\omega_y^* = \frac{\partial u^*}{\partial z^*} - \frac{\partial w^*}{\partial x^*} \tag{5.16}$$

which is plotted in Figure 5.20 for the baseline case at the midspan of the domain. As the jet penetrates into the cross-flow, the Kelvin-Helmholtz instability grows along the windward side and generates shear-layer vortices. These vortices convect downstream, grow in size, and eventually dissipate in the jet far-field.

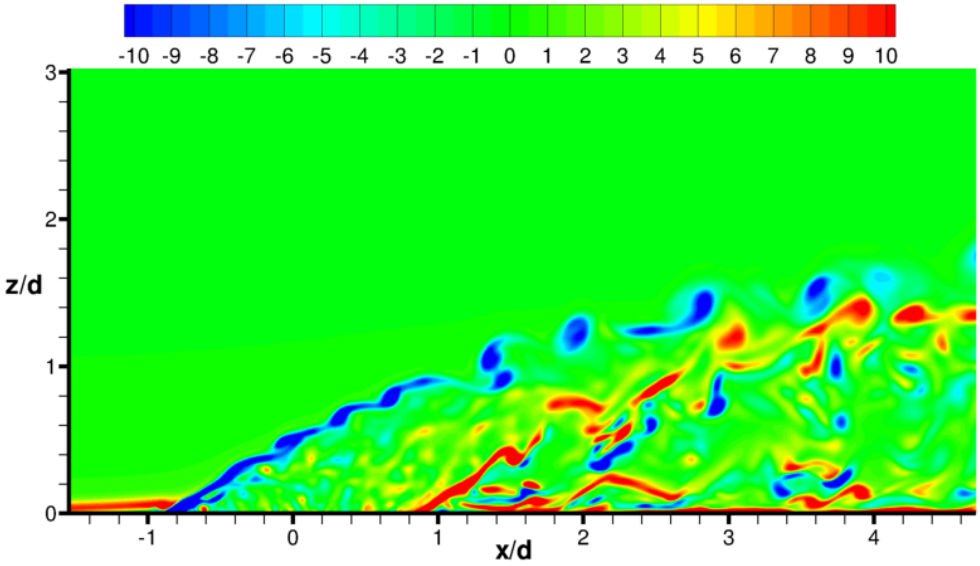


Figure 5.20: Instantaneous spanwise vorticity ω_y^* at midspan ($y/d = 0$) for baseline flow.

Visualization of the horseshoe vortex at the leading-edge of the jet at the midspan of the baseline flow is shown in Figure 5.21. Mean velocity vectors are used to show the roll-up of the cross-flow boundary layer as it impacts the jet flow. The streamwise location of the horseshoe vortex core is located at approximately $x/d = -0.9$. The mean temperature contours show

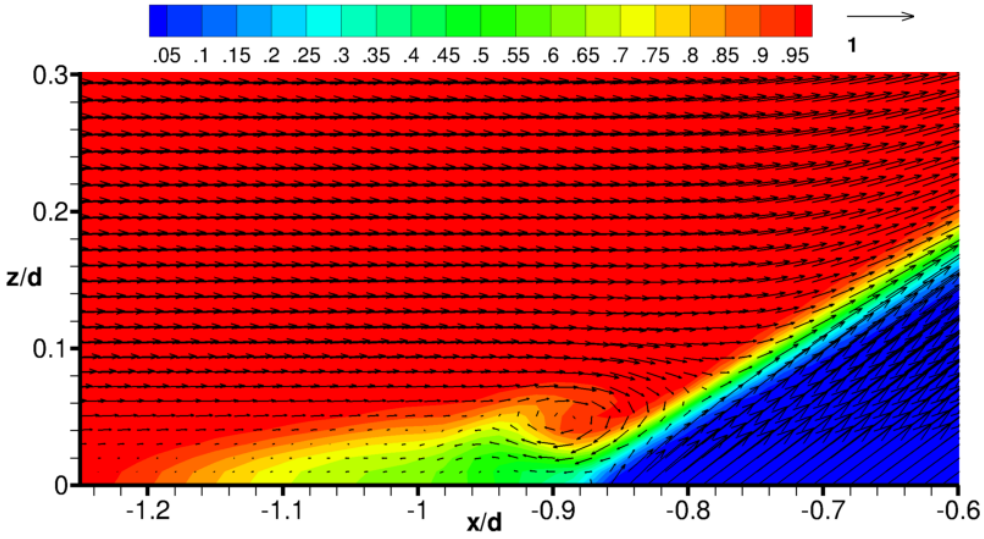
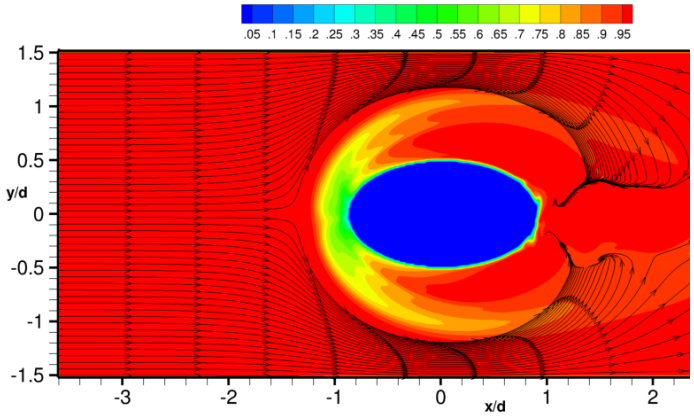


Figure 5.21: Horseshoe vortex near the jet leading-edge of baseline flow at midspan ($y/d = 0$), visualized using mean velocity vectors. Mean temperature contours indicate beneficial transport of coolant via the horseshoe vortex.

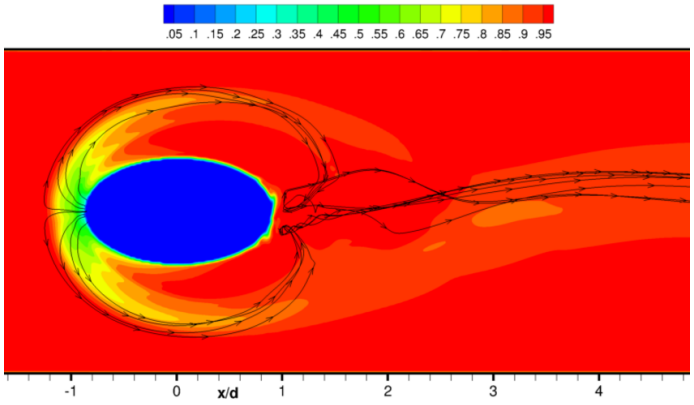
that the coolant is entrained by the horseshoe vortex and is transported upstream, creating an envelope of lower temperature fluid near the leading edge. This indicates that horseshoe vortices play a beneficial role in film-cooling.

Figure 5.22 shows streamlines and temperature contours from the mean field of the baseline flow near the jet. In Figure 5.22(a), streamlines are plotted along the wall starting from the inflow plane. These streamlines indicate that the flow moves around the base of the jet and converges on the leeward side of the jet owing to the low pressure region located there. In particular, we note that fluid becomes entrained in what appear to be symmetric vortices near the trailing-edge of the jet. A similar observation is made from Figure 5.22(b), where streamlines are plotted starting from the jet leading-edge and move to the jet trailing-edge and become entrained in vortical structures. The streamlines of Figure 5.22(b) are shown in three-dimensions in Figure 5.22(c), which shows that once the flow is entrained at the trailing-edge it is transported up into the jet along vortical structures.

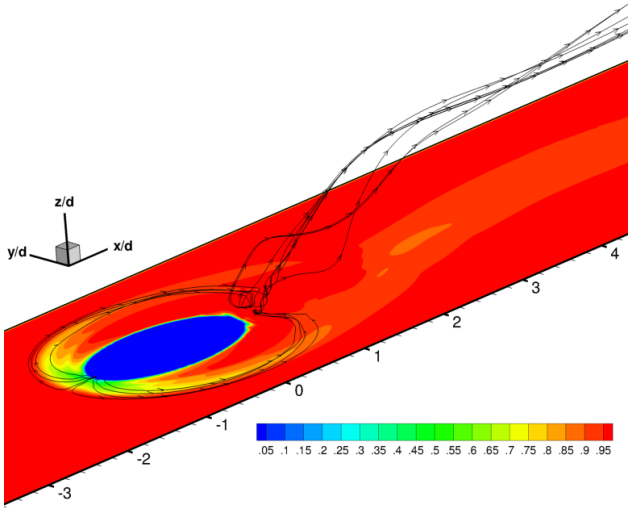
These observations are also found in the flow with a micro-ramp, as shown in Figure 5.23. These vortices are clearly visible in the mean flow and are therefore not "wake vor-



(a) Mean streamlines along wall near jet (top view)

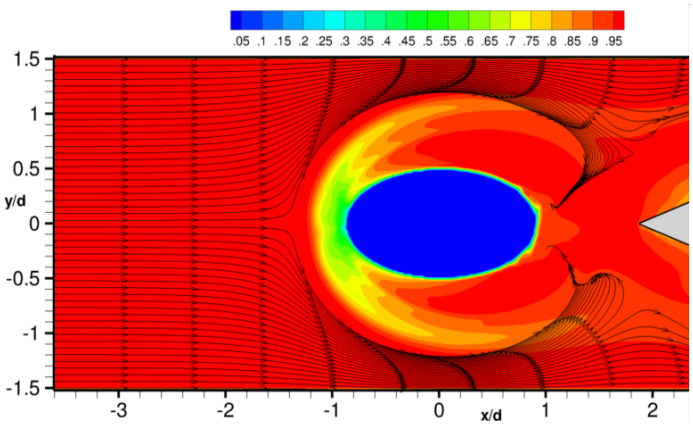


(b) Mean streamlines emanating from leading-edge of jet (top view)

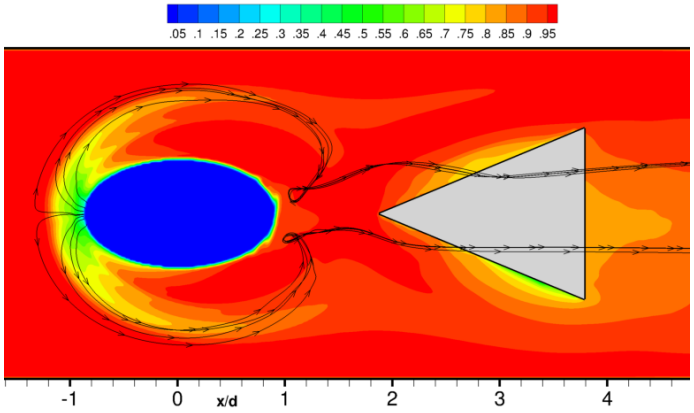


(c) Mean streamlines emanating from leading-edge of jet (three-dimensional view)

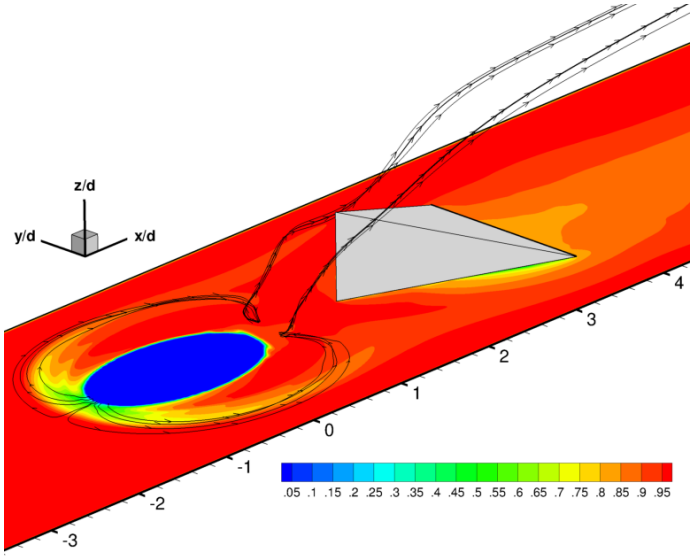
Figure 5.22: Mean streamlines for visualizing downstream spiral separated node (DSSN) vortices in baseline flow, superimposed on mean temperature contours.



(a) Mean streamlines along wall near jet (top view)



(b) Mean streamlines emanating from leading-edge of jet (top view)



(c) Mean streamlines emanating from leading-edge of jet (three-dimensional view)

Figure 5.23: Mean streamlines for visualizing downstream spiral separated node (DSSN) vortices in flow with micro-ramp, superimposed on mean temperature contours.

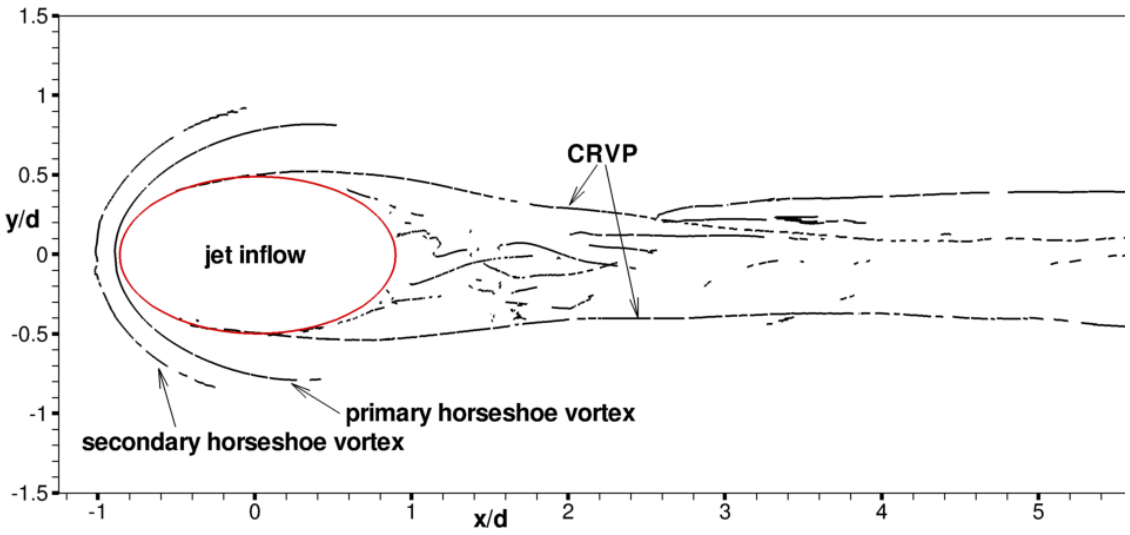
tices" that have been mentioned previously, since these are unsteady structures. These mean vortices downstream of the jet were observed experimentally by Peterson and Plesniak [12] and predicted numerically by Peet and Lele [32]. Peterson and Plesniak named these coherent structures "downstream spiral separation node (DSSN)" vortices, which will be the terminology adopted here. No unsteady wake vortices were observed in the present simulations, presumably due to the low blowing ratio.

Another method for visualizing coherent structures is to extract vortex cores. This can be done in a number of ways, such as using the vorticity vector or using the eigenvalues and eigenvectors of the mean velocity gradient tensor $\nabla\overline{\mathbf{u}}$. In the present work, the latter method was used [111]. Figure 5.24 shows vortex cores extracted from the mean velocity field for the baseline flow. At the leading edge of the jet, two horseshoe vortex cores are observed. The vortex core closest to the jet inflow is the "primary" horseshoe vortex that is created due to the roll-up of the cross-flow boundary layer. The vortex core farther away from the jet inflow is a weaker "secondary" horseshoe vortex, which is driven by the primary vortex by pushing fluid upstream against the boundary layer fluid moving downstream; this weak vortex is located at $x/d = -1$ in Figure 5.21 and is almost not visible. A DSSN vortex core is visible in Figure 5.24, showing that the vortex is normal to the wall initially, then bends in the streamwise direction and then moves into the jet. The CRVP of the jet is clearly visible in Figure 5.24, where it originates at the lateral sides of the jet inflow hole. Mutual vortex induction of the CRVP causes the jet cores to contract in the spanwise direction, as seen in the top view, and move vertically (jet lift-off) as seen in the side view.

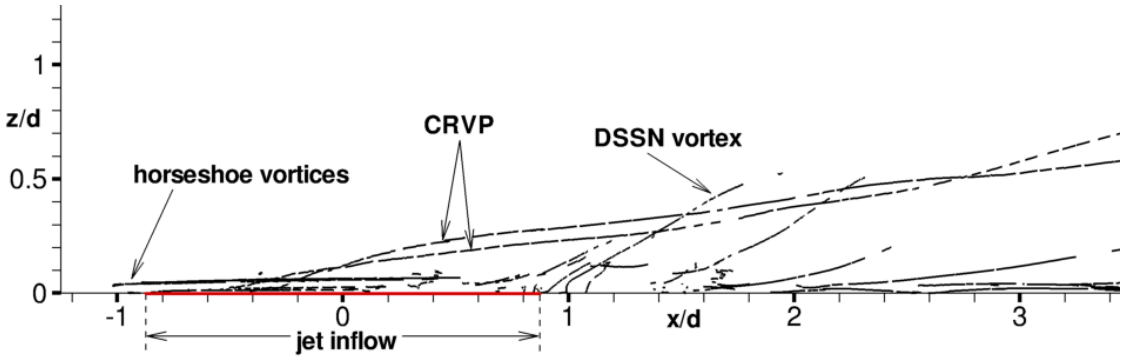
5.5.4 Film-Cooling Effectiveness

An important metric in the evaluation of a film-cooling design is the "film-cooling effectiveness" represented by η. This is computed as

$$\eta = \frac{(T_\infty - \overline{T}_w)}{(T_\infty - T_j)} = 1 - \overline{T}^*_w \tag{5.17}$$



(a) top view



(b) side view

Figure 5.24: Vortex cores extracted from mean flowfield for baseline flow. Jet inflow perimeter is highlighted in red.

where $\overline{T}_w$ is the mean wall temperature. Thus ideal cooling ($\overline{T}_w = T_j$) yields an effectiveness of unity, and no cooling ($\overline{T}_w = T_\infty$) yields an effectiveness of zero. Another useful metric is the span-averaged film-cooling effectiveness, computed as

$$\overline{\eta}(x) = \frac{1}{L_y} \int_{-L_y/2}^{L_y/2} \eta(x, y)\ dy \tag{5.18}$$

where L_y is the spanwise length of the domain. The film-cooling effectiveness at the wall is shown in Figure 5.25; note that from the definition in Equation (5.17) that the film-cooling effectiveness contours are the same as the mean temperature contours at the wall except the contour scale is reversed.

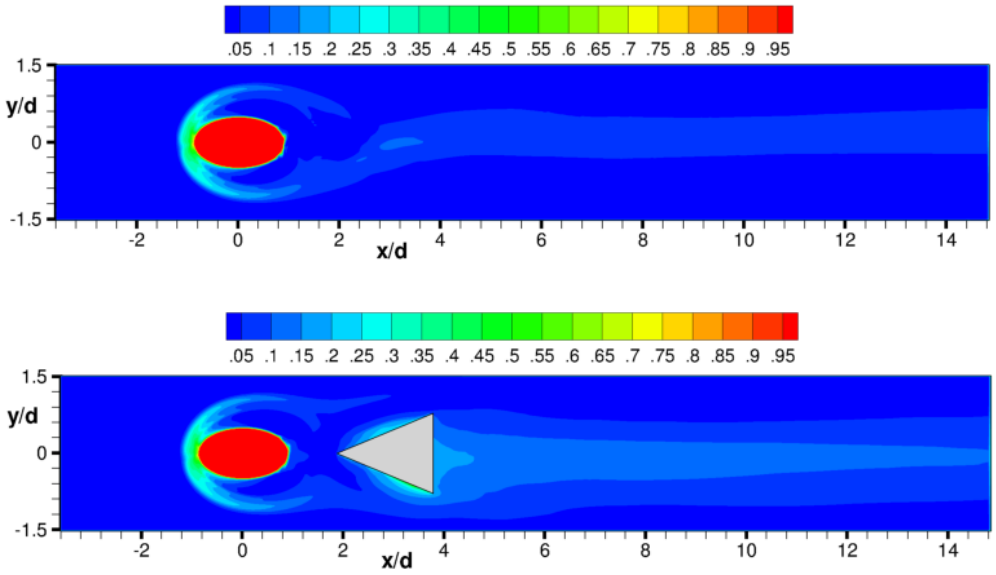


Figure 5.25: Contours of film-cooling effectiveness along wall of domain ($z/d = 0$).

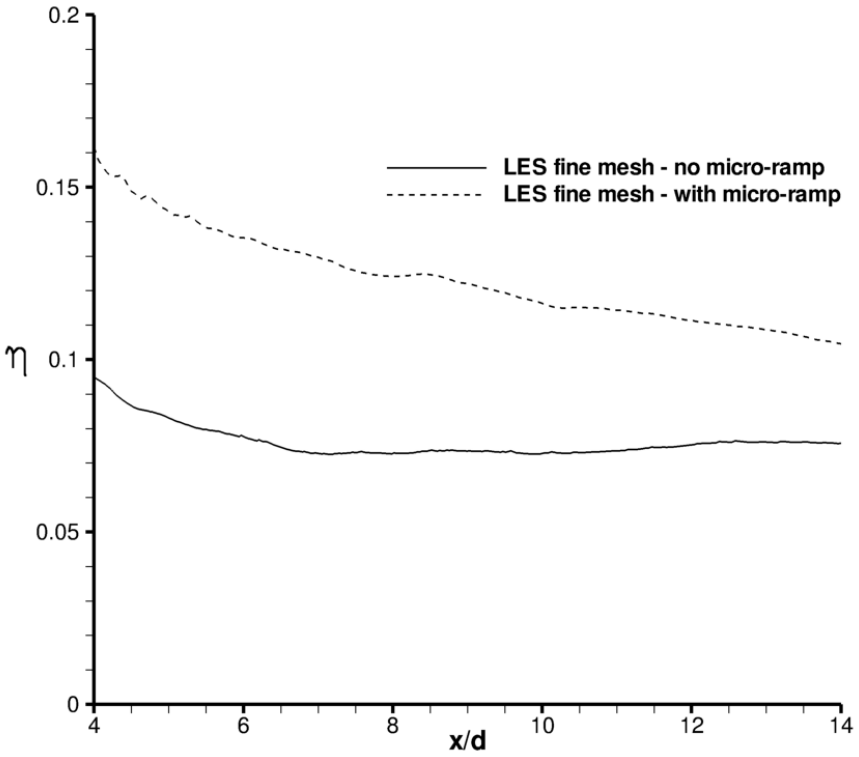
For both the baseline and the micro-ramp cases, the effectiveness is high near the leading edge of the jet, due to the entrainment and distribution of coolant by the horseshoe vortices, as explained earlier. In the downstream region, the effectiveness is higher for the micro-ramp case, especially near the centerline where the effect of the downwash from the micro-ramp vortices is largest. As we move away from the centerline, we see the effectiveness drop, eventually to zero. The centerline film-cooling effectiveness and span-averaged film-cooling

effectiveness are plotted in Figure 5.26 for the region downstream of the jet and micro-ramp. The micro-ramp shows improvement compared with the baseline case for the entire downstream region, but we note that the margin of improvement decreases in the streamwise direction.

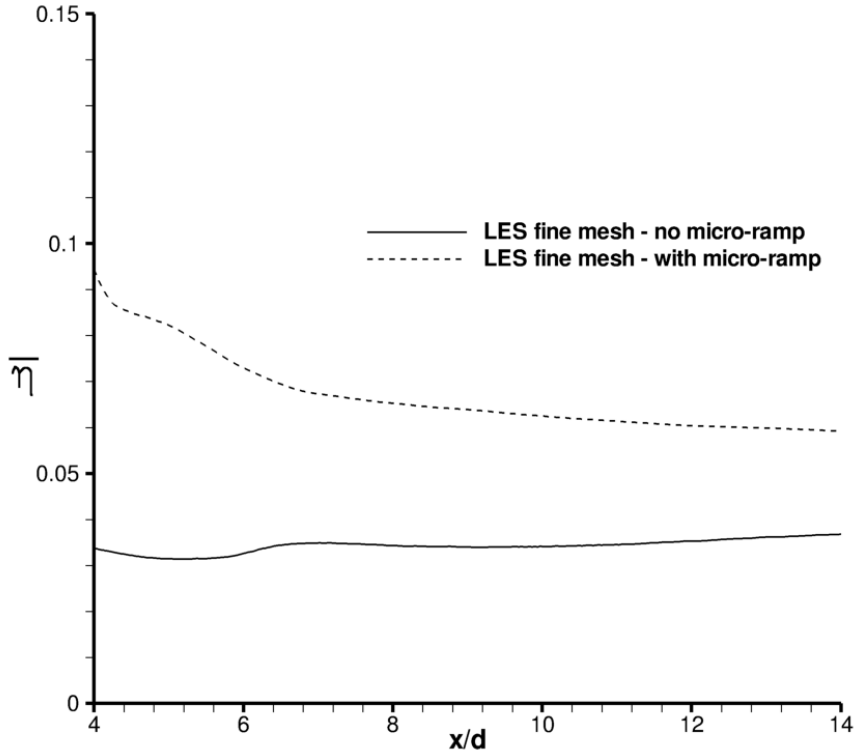
For validation purposes, centerline and span-averaged effectiveness from the present baseline case are compared to the DNS of Muldoon and Acharya [19] in Figure 5.27. The region shown is around the jet (centered at $x/d = 0$) and downstream of the jet. For the centerline effectiveness, the downstream predictions agree very well with [19], but there is disagreement in the jet near-field. For the span-averaged effectiveness, we again see disagreement occurs in the jet near-field and also in the jet far-field. The disagreement in the near-field is not surprising, as this region has a very complex flow owing to the cross-flow moving around both sides of the jet and then colliding near the trailing edge. In addition, it should be noted that the prescription of the jet inflow velocity boundary condition was different between the present study and that of [19], which could cause changes in the results. The present study used an unsteady velocity boundary condition (as described earlier), whereas the previous study of [19] used time-averaged results from a precursor simulation. The plots in Figure 5.27 also compare the predicted effectiveness between the coarse and fine mesh simulations. Overall, reasonable agreement is observed, indicating that the results are nearly grid independent. Since a computationally cheaper mesh can provide nearly the same prediction of film-cooling performance, quick and accurate design calculations using LES can be performed, which is very attractive from a practical perspective.

5.6 Conclusions and Recommendations

In this chapter, Large Eddy Simulations were presented to study an inclined turbulent jet interacting with a cross-flow in a film-cooling configuration at low Reynolds number. Instead of modeling the plenum and coolant pipe, a precursor simulation was used to create

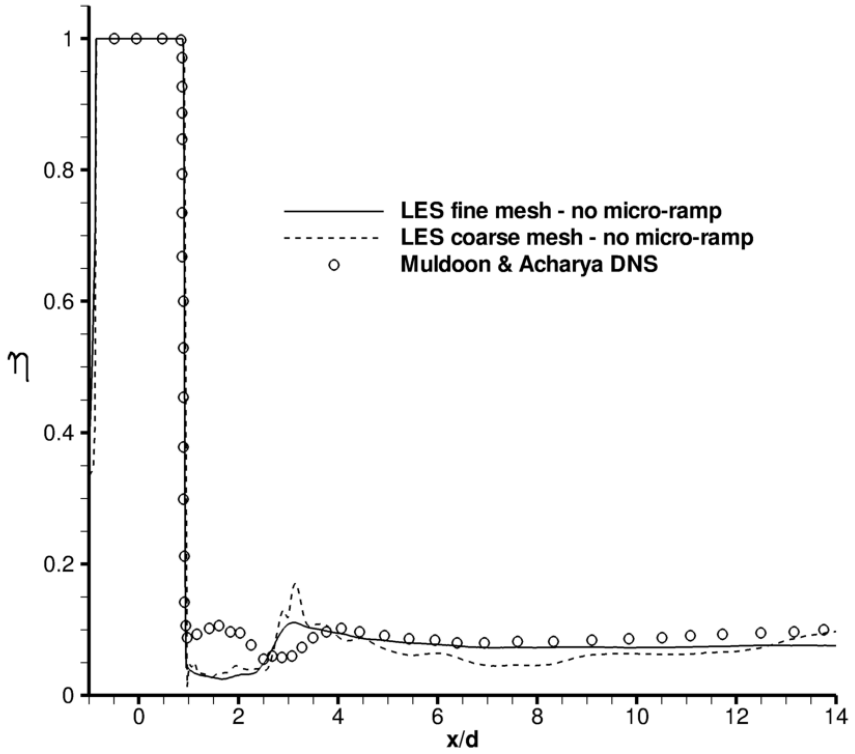


(a) film-cooling effectiveness along centerline of domain

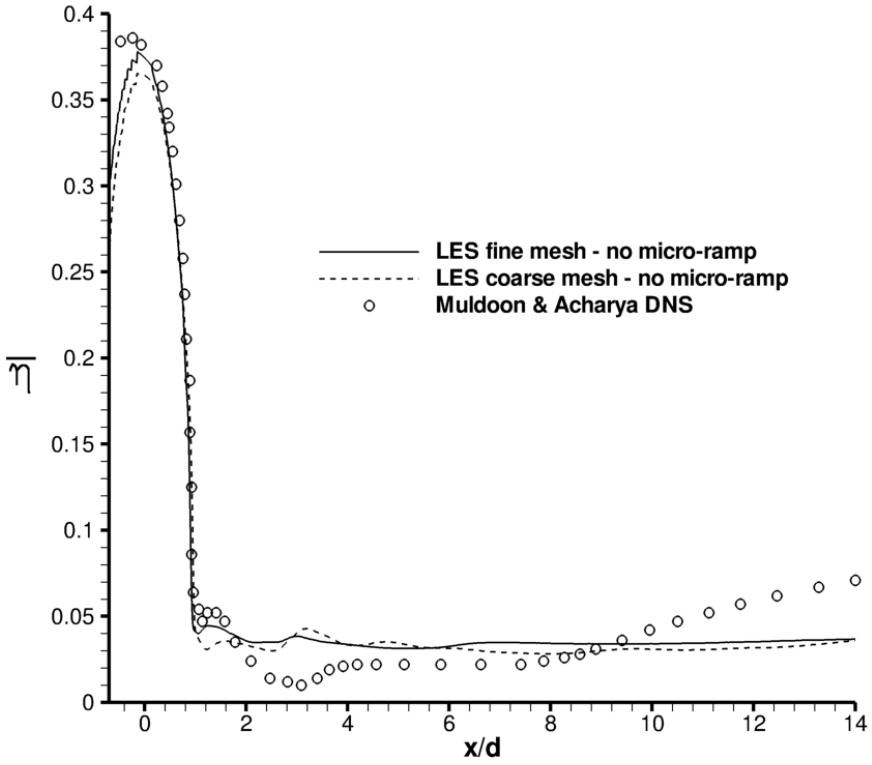


(b) span-averaged film-cooling effectiveness

Figure 5.26: Film-cooling effectiveness comparison between baseline flow and flow with micro-ramp.



(a) film-cooling effectiveness along centerline of domain



(b) span-averaged film-cooling effectiveness

Figure 5.27: Predicted film-cooling effectiveness compared with previous DNS data from Muldoon and Acharya [19].

an unsteady velocity boundary condition for the jet inflow. The purpose of this study was to compare how a micro-ramp vortex generator placed downstream of the jet alters the flowfield compared to a baseline case with no micro-ramp and to quantify any increase in cooling performance using the micro-ramp. Simulations were performed using coarse and fine meshes to assess the effect of resolution on film-cooling predictions. Results for instantaneous and time-averaged velocity and temperature fields were presented. Coherent structures in the flowfield were identified and discussed. The following summarizes the conclusions of this chapter:

1. Coherent structures were identified, including shear-layer vortices on the windward side of the jet, two horseshoe vortices at the jet leading-edge (primary vortex driving a weaker secondary vortex), a counter-rotating vortex pair (CRVP) in the jet, and downstream spiral separation node (DSSN) vortices. No wake vortices were found, possibly due to the low blowing ratio.

2. The horseshoe vortex was found to the very beneficial in cooling the wall near the jet leading edge due to entrainment and distribution of jet coolant.

3. For the baseline case, the jet CRVP was the dominant flow structure affecting film-cooling in the jet far-field. It was not visible in the instantaneous flow but became very clear in the mean flow. Vortex induction by the CRVP caused the jet to lift away from the surface. The vortices were rotating such that an upwash region was generated between the vortices. This upwash caused the coolant to be moved away from the wall and simultaneously entrained hot cross-flow and transported it toward the wall, resulting in reduced cooling effectiveness.

4. The micro-ramp generated a pair of counter-rotating vortices of opposite sense to the CRVP in the jet. The micro-ramp vortices helped weaken the jet CRVP through vorticity cancellation and created a downwash effect that entrained and transported coolant

from the jet toward the wall. The latter effect enhanced film-cooling effectiveness at the wall compared to the baseline case.

5. The vortices generated by the micro-ramp were weak and decayed quickly in the streamwise direction; however, the beneficial downwash effect induced by these vortices persisted downstream. The angle of inclination of the jet was such that the jet flow moved over the micro-ramp, which left only a low-speed flow in the jet wake to move past the micro-ramp. This low-speed flow was responsible for generating the weak vortex pair as it moved past the micro-ramp.

6. Adding a micro-ramp did not appreciably alter the jet trajectory, and jet lift-off was still observed. Despite this, the effect of creating a downwash near the wall was sufficient to improve film-cooling effectiveness.

7. It was found that the mean velocity profile at the jet inflow was not symmetric, due to the jet inflow data being recycled, resulting in a smaller sample size. Thus, it is recommended for future studies to either store more precursor data or run the precursor simulation in parallel with the main simulation and communicate the jet inflow data every time-step; either way, the statistics in the jet inflow will improve. Of course, the best way to construct this simulation is to model the plenum and coolant pipe in the computational domain, which will correct the asymmetry problem and allow the jet profile to adjust to the effect of the cross-flow. This approach is employed in the next chapter.

8. Initially, the film-cooling problems of this chapter were simulated using "unresolved" DNS, where no sub-grid scale model was used but the dissipative scales were not resolved by the mesh. This method resulted in numerical divergence of the solution and was probably caused by a pile-up of turbulent kinetic energy. Using LES was stable, since the SGS model helps transfer the turbulent kinetic energy to smaller

scales.

9. It is recommended that future studies examine alternative placements of the micro-ramp (for example, upstream of the jet inflow) in an effort to find an optimal configuration.

Chapter 6

LES of Film-Cooling Flow with a Micro-Ramp Vortex Generator: Jet Modeled with Plenum and Pipe

6.1 Introduction

Large Eddy Simulations are performed to study the flow in a film-cooling configuration with a micro-ramp vortex generator placed downstream of the film-cooling hole. This configuration is based on windtunnel experiments conducted at NASA by Zaman et al. [23], where they experimentally studied the interaction of an inclined jet in a cross-flow and a micro-ramp vortex generator. This numerical study is meant to supplement those experiments by providing a three-dimensional, time-accurate representation of the flow. In particular, information about the film-cooling effectiveness and temperature field are provided, which were not studied in the experiments. The present simulations create the jet exit conditions by including a plenum chamber and jet pipe upstream of the jet exit, unlike in the previous chapter where a precursor simulation was used for the jet exit. The geometry and boundary conditions were prescribed to match the experimental conditions as closely as possible.

The purpose of this study is to compare how a micro-ramp vortex generator placed downstream of a film-cooling jet alters the flow field characteristics compared to a baseline case with no micro-ramp. Of particular interest is the temperature distribution created by the two configurations and the corresponding film-cooling effectiveness. Two simulations are performed: an LES of an inclined coolant jet in a hot cross-flow, where the cross-flow moves across a flat plate (baseline case) and another LES with the same geometry and boundary conditions except a micro-ramp is placed one diameter downstream of the jet hole. Comparison with experiments in [23] will be made to assess the accuracy of the simulations.

This chapter is organized as follows: first, a brief description of the previous NASA experiments will be given. Next, the problem description is given along with the numerical methods and boundary conditions. Then results are presented including instantaneous and time-averaged velocity and temperature fields, along with a comparison with experimental data. Results for film-cooling effectiveness will be shown to quantify the improvement delivered by the micro-ramp. Lastly, the chapter closes with conclusions and future work.

6.2 Description of Recent Experiments

Recently, experiments have been performed at NASA Glenn Research Center by Zaman, Rigby, and Heidmann [23] to explore the use of vortex generators interacting with an inclined jet in a cross-flow. The motivation was to see if a micro-ramp vortex generator could prevent jet lift-off, which could have beneficial consequences in film-cooling flows. They performed experiments using an open-loop wind tunnel with a test section that was 20 inches high and 30 inches wide. In the test section, a single micro-ramp vortex generator was placed downstream of a single inclined jet, as shown in Figure 6.1. The jet was inclined at an angle of 20 degrees with respect to the test section floor and the jet diameter was $d = 0.75$ inches. The jet hole was bored through a 1 inch thick plate. A hose supplying shop air was connected to a flow conditioning screen that led into a wooden box that acted as a plenum. The plenum box was located below the jet (which can be seen through the transparent jet plate in Figure 6.1). Air from the plenum flowed into the jet tube and was injected into the cross-flow at the tunnel floor. The experiments were performed at a Reynolds number of 11,400 based on the freestream velocity and jet exit diameter. The free-stream velocity was $u_\infty = 29\ ft/s$ and the bulk jet velocity was $u_j = 41\ ft/s$. Thus the blowing ratio was $u_j/u_\infty \approx \sqrt{2}$.

Hot-wire anemometry was used to measure mean velocity, r.m.s. velocity, and streamwise vorticity. A single hot-wire was used for measurements at the jet exit and two X-wires were

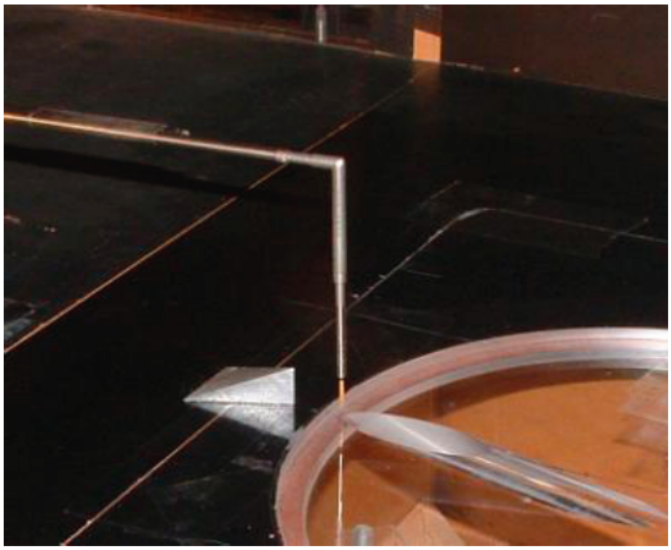


Figure 6.1: Micro-ramp downstream of an inclined jet inside wind tunnel test section at NASA Glenn Research Center [23].

used for measurements in streamwise planes downstream of the jet and micro-ramp. Surveys at various streamwise locations were performed for the H0(R0) micro-ramp model located one diameter downstream of the jet. It was found that the vorticity generated by the micro-ramp canceled-out the vorticity in the jet, leaving only the counter-rotating vortices from the micro-ramp. As a result, the trajectory of the jet moved closer to the wall, indicating that this configuration could be beneficial for film-cooling. However, their study was not designed as a film-cooling flow, so no temperature data were taken and therefore cooling effectiveness was not assessed.

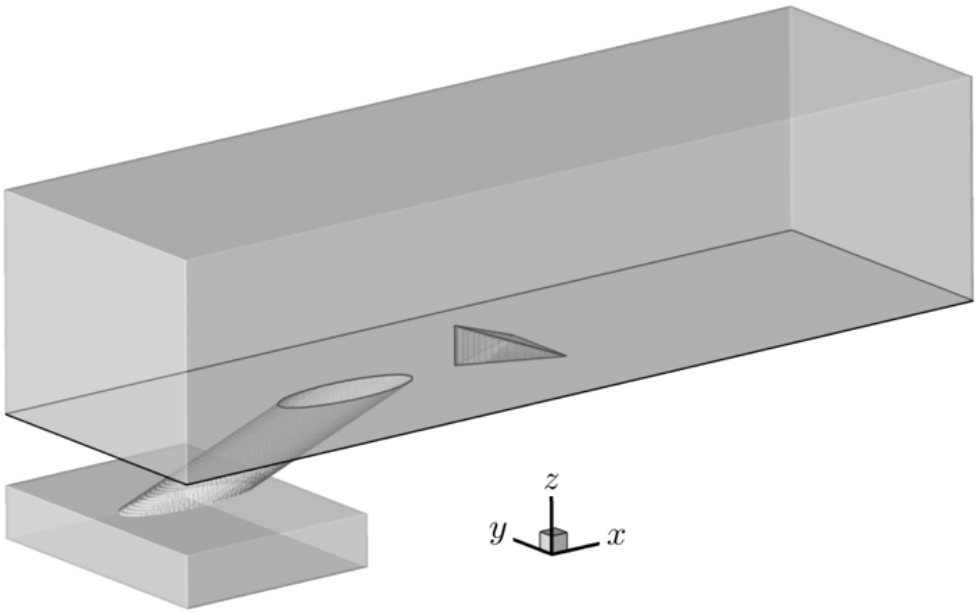
They also investigated varying the micro-ramp height, micro-ramp location, and micro-ramp edge curvature. It was observed that when the micro-ramp height is halved, the jet lifts off the wall and if the micro-ramp height is doubled, the jet is dissipated owing to an increase in turbulence intensity. They found that moving the micro-ramp over a distance of three diameters from the jet exit did not appreciably affect the results. Rounding-off the micro-ramp edges was found to weaken the counter-rotating vorticies leading to a decrease in effectiveness.

6.3 Description of Computational Model

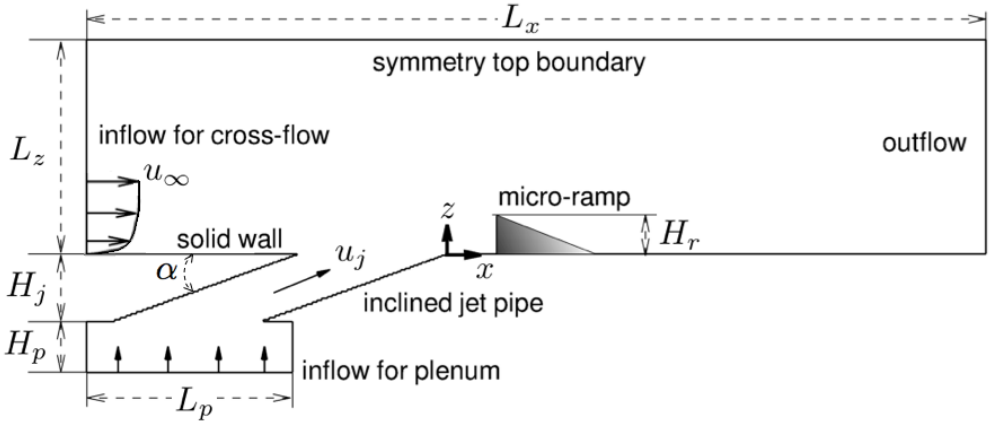
The computational domain is shown in Figure 6.2 and the corresponding dimensions are given in Table 6.1. The coolant enters a plenum chamber vertically, and is then directed into an inclined circular pipe, where the angle of inclination is 20 degrees relative to the flat plate surface. The jet of coolant then enters into the hot cross-flow, where the cross-flow is a boundary-layer moving across a flat plate. The interaction of the jet and the cross-flow creates a complex, highly three-dimensional turbulent flow with strong fluid mixing that persists downstream to the outflow. The geometry of the micro-ramp used in the present simulations is based on the H0(R0) model used in the experimental study by Zaman et al. [23]. The blowing ratio (u_j/u_∞) was approximately $\sqrt{2}$ and the Reynolds number based on the jet diameter and freestream cross-flow velocity was $Re_\infty = u_\infty d/\nu = 11,400$. The plenum velocity was set such that the desired blowing ratio was achieved (this will be discussed in detail in section 6.5).

Table 6.1: Dimensions of computational domain.

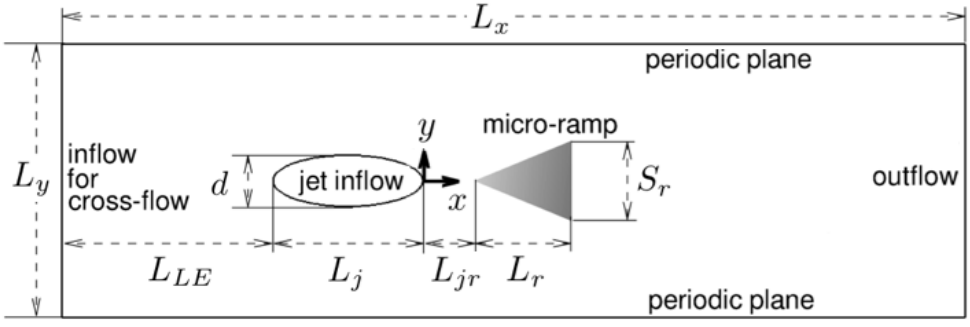
L_x	$17.71d$
L_y	$5.35d$
L_z	$4.21d$
L_{LE}	$4.15d$
L_j	$2.933d$
L_{jr}	$1.0d$
L_r	$1.913d$
S_r	$1.573d$
H_r	$0.748d$
H_j	$1.333d$
H_p	$1.0d$
L_p	$4.05d$
α	20 degrees



(a) three-dimensional view



(b) side view



(c) top view

Figure 6.2: Computational domain for film-cooling configuration with a micro-ramp.

6.4 Numerical Methodology

Large Eddy Simulations were performed using the GPU-based flow solver CU-FLOW that was described in Chapter 3. The WALE eddy viscosity model was used with a constant of $C_w = 0.55$. The LES equations were non-dimensionalized using the jet exit diameter d and the freestream velocity u_∞. The non-dimensional temperature is defined as $T^* = (T - T_j)/(T_\infty - T_j)$, where T_∞ is the freestream temperature of the cross-flow (assumed constant) and T_j is the jet temperature (assumed constant). The temperature is one-way coupled with the velocity field and is thus treated as a passive conserved scalar. The Prandtl number was taken as 0.7, representative of air.

A Cartesian mesh with non-uniform spacing was used for the LES. In the region above the flat plate, the mesh resolution was 576 cells (streamwise direction) x 160 cells (spanwise direction) x 100 cells (vertical direction) $\approx$ 9.2 million cells. The jet inflow area was meshed using 128 cells along the major axis and 64 cells along the minor axis. The mesh spacings in the jet inflow area were uniform such that $\Delta x_j = L_j/128$ and $\Delta y_j = d/64$. The streamwise mesh spacings downstream of the jet and spanwise mesh spacings were set by stretching geometrically from the jet inflow area. A boundary layer mesh was generated at the wall with 30 cells inside of a vertical height of $\delta = d$, where the first cell at the wall had a spacing of $\Delta z/d \approx 0.029$. The jet pipe was meshed using 40 cells in the vertical direction and the surface is represented using a stair-step approximation in the mesh. The plenum was meshed using 176 cells (streamwise direction) x 160 cells (spanwise direction) x 20 cells (vertical direction). A ghost cell immersed boundary method (see Chapter 3) was used for the micro-ramp surface, where three segments that are coincident with the micro-ramp faces were used for the surface triangulation.

Since the cell Peclet number is very high for these simulations, a flux limiter was used for the convective term to avoid spurious oscillations from central differencing. The flux limiter used was van Leer's monotonized central (MC) limiter [112, 113]. The location of the first

u-velocity off from the wall was at a wall-normal distance of $z/d \approx 0.014$, which in wall units is $z^+ \approx 14.35$ and is located in the buffer layer of a turbulent flat plate boundary-layer. Since the laminar sublayer was not resolved, it was decided to try wall functions along the flat plate surface in an attempt to better represent the wall shear stress. The wall function proposed by Werner and Wengle [114] was used to compute the wall shear stress as

$$|\tau_w| = \begin{cases} \frac{2\mu|u_1|}{\Delta z} & \text{for} \quad |u_1| \leq \frac{\mu}{2\rho\Delta z} A^{\frac{2}{1-B}} \\ \rho \left[\frac{1-B}{2} A^{\frac{1+B}{1-B}} \left(\frac{\mu}{\rho\Delta z} \right)^{1+B} + \frac{1+B}{A} \left(\frac{\mu}{\rho\Delta z} \right)^{B} |u_1| \right]^{\frac{2}{1+B}} & \text{for} \quad |u_1| > \frac{\mu}{2\rho\Delta z} A^{\frac{2}{1-B}} \end{cases} \tag{6.1}$$

where u_1 is the first streamwise velocity from the wall, Δz is the height of the first cell adjacent to the wall, $A = 8.3$, and $B = 1/7$. Note that this wall function can be applied to any velocity component parallel to a wall; thus it is also applied to the spanwise velocity using v_1. Since the present problem is solved non-dimensionally, we need to non-dimensionalize Equation (6.1) using d and u_∞, which yields

$$|\tau_w^*| = \begin{cases} 2\frac{1}{Re_\infty}\frac{|u_1^*|}{\Delta z^*} & \text{for} \quad |u_1^*| \leq \frac{1}{2\Delta z^*}\frac{1}{Re_\infty} A^{\frac{2}{1-B}} \\ \left[\frac{1-B}{2} A^{\frac{1+B}{1-B}} \left(\frac{1}{Re_\infty}\frac{1}{\Delta z^*} \right)^{1+B} + \frac{1+B}{A} \left(\frac{1}{Re_\infty}\frac{1}{\Delta z^*} \right)^{B} |u_1^*| \right]^{\frac{2}{1+B}} & \text{for} \quad |u_1^*| > \frac{1}{2\Delta z^*}\frac{1}{Re_\infty} A^{\frac{2}{1-B}} \end{cases} \tag{6.2}$$

Once the shear stress is computed from Equation (6.2) it is multiplied by the wall area over which it acts to get the force at the wall

$$F_w^* = \pm \tau_w^* A_w^* \tag{6.3}$$

where the sign of the shear stress is taken to be the opposite of the sign of the velocity component u_1^* (or opposite of v_1^* if applying to the v-velocity). The area is $A_w^* = 0.5(\Delta x_i^* + \Delta x_{i+1}^*)\Delta y_j^*$ if applying the wall function to u_1 or $A_w^* = 0.5(\Delta y_j^* + \Delta y_{j+1}^*)\Delta x_i^*$ if applying the wall function to v_1. The diffusion term D_u (or D_v) for the cell adjacent to the wall is then modified by setting the numerical diffusion flux for the wall to zero ($\widehat{F}_{z-}^d = 0$) and in

its place adding the wall force F_w^* (cf. Appendix A). Thus the diffusion term is adjusted by the wall function to yield a better approximation of the wall shear stress for a turbulent boundary-layer.

The time-step used for the simulations was $\Delta t = 0.001(d/u_\infty)$, which was selected to ensure temporal accuracy and satisfy the CFL condition, a restriction imposed by the explicit nature of the algorithm. The problem was simulated for a total of $400(d/u_\infty)$ time units, and mean statistics were collected starting at $20(d/u_\infty)$ time units and continued until the end of the simulation. Using a Tesla C1060 GPU, the LES of the flow with no micro-ramp took 4.5 $seconds/\Delta t$ and the LES of the flow with a micro-ramp took 5.7 $seconds/\Delta t$.

6.5 Boundary Conditions

6.5.1 Inflow Boundary for Cross-Flow

The inflow boundary condition for the cross-flow was modeled after the experimental study. Figure 6.3 shows a comparison between the experimentally measured mean boundary-layer profile (at $x/d = -2.95$, just upstream of the jet hole) and a 1/7 power law, given as

$$\frac{\overline{u}}{u_\infty} = \left(\frac{z}{\delta}\right)^{1/7} \tag{6.4}$$

with boundary layer height of $\delta = 0.6d$, which was the experimentally measured height at the $x/d = -2.95$ location. It can be seen that the 1/7 power law provides a reasonable approximation, as expected. Thus it is reasonable to use a 1/7 power law as the mean inflow profile. To do this, the boundary layer height at the inflow at $x/d = -7.08$ is needed, which can be deduced using the known boundary layer height of $\delta = 0.6d$ at $x/d = -2.95$ and the well-known relation

$$\frac{\delta}{x} \approx \frac{0.16}{Re_x^{1/7}} \tag{6.5}$$

where the Reynolds number $Re_x = u_\infty x/\nu$ is based on the streamwise distance along a flat plate (here called x) [115]. This yields a boundary layer height at the inflow of $\delta = 0.5028d$ which is substituted into the power law above to produce a profile for the mean streamwise velocity $\overline{u}^*$ for the cross-flow. The mean spanwise and vertical components were set to zero ($\overline{v}^* = \overline{w}^* = 0$).

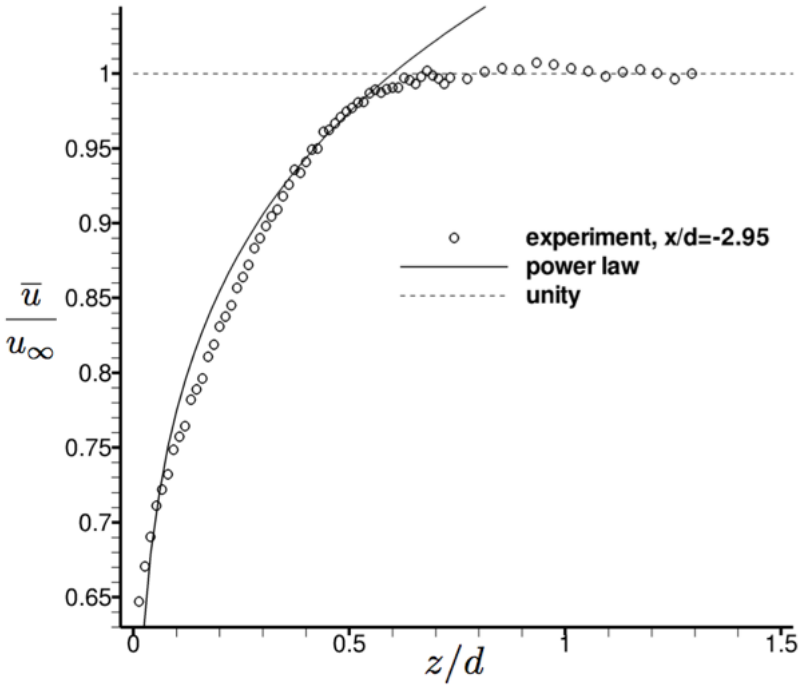


Figure 6.3: Comparison of experimentally measured mean boundary-layer profile (at $x/d = -2.95$) and a 1/7 power law.

In an LES, the instantaneous velocity field must be specified at the inflow. Unsteadiness at the inflow due to turbulent fluctuations can be represented by first decomposing the non-dimensional instantaneous velocity components into mean and fluctuating velocities as

$$u^* = \overline{u}^* + u'^* \tag{6.6}$$

$$v^* = \overline{v}^* + v'^* \tag{6.7}$$

$$w^* = \overline{w}^* + w'^* \tag{6.8}$$

We assume that $\overline{v}^* = \overline{w}^* = 0$ and that the fluctuations are isotropic ($u'^* = v'^* = w'^*$) at the inflow, thus

$$u^* = \overline{u}^* + u'^* \tag{6.9}$$

$$v^* = u'^* \tag{6.10}$$

$$w^* = u'^* \tag{6.11}$$

The mean streamwise velocity $\overline{u}^*$ at the inflow is prescribed using the power law described earlier. The streamwise fluctuating velocity u'^* at the inflow can be modeled using the turbulence intensity I, a pseudo-random number ψ, and the mean streamwise velocity as

$$u'^* = I\psi\overline{u}^* \tag{6.12}$$

where the turbulence intensity is defined as the root-mean-square of the streamwise velocity fluctuation divided by the local average velocity

$$I = \frac{\sqrt{\overline{(u'^*)^2}}}{u^*_{avg}} = \frac{u^*_{rms}}{u^*_{avg}}. \tag{6.13}$$

The random number ψ is a Gaussian random number with a standard-normal distribution. Thus ψ has a zero mean, $\overline{\psi} = 0$, and a unity variance, $\overline{\psi'^2} = 1$. The Gaussian random numbers ψ are generated on the GPU by using a uniform pseudo-random number generator and then transforming the uniform random numbers to a Gaussian distribution. The uniform random number generator is a hybrid LCG/Tausworthe generator and the Gaussian transformation is performed using the Box-Muller transform [116].

For the turbulence intensity, we take the local average velocity to be the local mean velocity ($u^*_{avg} = \overline{u}^*$), thus

$$I = \frac{u^*_{rms}}{\overline{u}^*} \tag{6.14}$$

Substituting Equation (6.14) into Equation (6.12) yields

$$u'^* = u^*_{rms}\psi \tag{6.15}$$

Substituting Equation (6.15) into Equations (6.9) to (6.11) gives the final form of the inflow boundary condition for the cross-flow

$$u^* = \overline{u}^* + u^*_{rms}\psi \tag{6.16}$$

$$v^* = u^*_{rms}\psi \tag{6.17}$$

$$w^* = u^*_{rms}\psi \tag{6.18}$$

The mean profile $\overline{u}^* = \overline{u}^*(z^*)$ is specified using a 1/7 power law, described earlier. The r.m.s. profile $u^*_{rms} = u^*_{rms}(z^*)$ is specified using a polynomial curve fit of the experimental data from [23] for the turbulent inflow boundary layer. The only rms data available was at $x/d = -2.95$, which was not at the inflow plane; thus this represents the best available approximation. The hot cross-flow temperature was prescribed as a uniform temperature T_∞ at the inflow (non-dimensionally $T^* = 1$).

6.5.2 Inflow Boundary for Plenum

The velocity boundary condition at the inflow to the plenum is constructed in a manner similar to the cross-flow described above. We start with the instantaneous velocities decomposed into mean and fluctuating components

$$u^* = \overline{u}^* + u'^* \tag{6.19}$$

$$v^* = \overline{v}^* + v'^* \tag{6.20}$$

$$w^* = \overline{w}^* + w'^* \tag{6.21}$$

At the plenum inflow we assume that only the mean vertical velocity is non-zero, thus $\overline{u}^* = \overline{v}^* = 0$. Also, we assume the fluctuations are isotropic ($u'^* = v'^* = w'^*$). Substituting these assumptions into Equations (6.19) to (6.21) yields

$$u^* = w'^* \tag{6.22}$$

$$v^* = w'^* \tag{6.23}$$

$$w^* = \overline{w}^* + w'^* \tag{6.24}$$

The fluctuating component is modeled using

$$w'^* = I\psi\overline{w}^* \tag{6.25}$$

which was introduced earlier, where again I is the turbulence intensity and ψ is a Gaussian pseudo-random number. Substituting Equation (6.25) into Equations (6.22) to (6.24) gives the final form of the plenum inflow boundary condition for instantaneous velocity:

$$u^* = I\psi\overline{w}^* \tag{6.26}$$

$$v^* = I\psi\overline{w}^* \tag{6.27}$$

$$w^* = \overline{w}^* + I\psi\overline{w}^* \tag{6.28}$$

The mean vertical velocity at the plenum inflow ($\overline{w}^*$) was derived from continuity, since the volume flow rate through the plenum inflow must be equal to the volume flow rate through the jet pipe cross-section. This is derived as follows using non-dimensional variables

$$Q_p^* = Q_j^* \tag{6.29}$$

$$V_p^* A_p^* = V_j^* A_j^* \tag{6.30}$$

where the subscript p indicates the plenum inflow, j indicates the jet, V_p^* is the plenum velocity and is equal to the mean vertical velocity $\overline{w}^*$ at the plenum inflow, V_j^* is the jet velocity and is equal to the blowing ratio $u_j/u_\infty = \sqrt{2}$, the plenum inflow area is $A_p^* = L_p^* L_y^* \approx 21.7$, and the cross-sectional area of the jet pipe is $A_j^* = \pi/4$. Thus,

$$\overline{w}^* = \frac{(u_j/u_\infty)A_j^*}{A_p^*} = \frac{(\sqrt{2})(\pi/4)}{21.7} \approx 0.051 \tag{6.31}$$

In the experiments, no measurements were taken for the plenum conditions, so little is known about the turbulence statistics there. It is assumed that the turbulence levels are low, so a uniform turbulence intensity of $I = 1\% = 0.01$ was used. The instantaneous temperature at the plenum inflow was set to the uniform temperature $T^* = 0$, which represents the coolant flow.

6.5.3 Top Boundary

At the top boundary, a symmetry boundary condition is used, representing a frictionless wall. For the velocities this is written as

$$\frac{\partial u^*}{\partial z^*} = \frac{\partial v^*}{\partial z^*} = w^* = 0 \tag{6.32}$$

and for the temperature as

$$\frac{\partial T^*}{\partial z^*} = 0 \tag{6.33}$$

where the gradient was implemented numerically using a first-order accurate one-sided difference.

6.5.4 Side Boundaries

For the side boundaries, a periodic boundary condition was used and was applied to all variables in the solution (u, v, w, T, p). While this boundary condition simulates the effect of having a spanwise row of film-cooling jets that are parallel to one another, it is believed that the domain is wide enough to avoid any interference with the jet development using this boundary condition. Thus this boundary condition is appropriate to use for modeling a single jet and micro-ramp in a wide domain, as in the present case.

6.5.5 Solid Boundaries

At all solid boundaries (flat plate, micro-ramp surface, pipe walls, and plenum walls) the no-slip condition is applied for velocity,

$$u^* = v^* = w^* = 0 \tag{6.34}$$

For the flat plate and micro-ramp an adiabatic boundary condition is used for temperature,

$$\frac{\partial T^*}{\partial n} = 0 \tag{6.35}$$

where n is the local wall-normal coordinate. In the case of the flat plate wall $n = z^*$ and for the micro-ramp surface n is along the surface normal of each triangular segment. For the pipe and plenum walls, a Dirchlet temperature boundary condition is employed by setting $T^* = 0$. This ensures that the jet remains "cold" until it enters the cross-flow region.

6.5.6 Outflow Boundary

At the outflow boundary a convective boundary condition is used on all velocity components and temperature:

$$\frac{\partial \phi}{\partial t} + u_c \frac{\partial \phi}{\partial x} = 0 \tag{6.36}$$

where ϕ is taken as u, v, w or T. The implementation of this boundary condition was described in detail in Chapter 5.

6.6 Results

6.6.1 Instantaneous Field

The following results for the instantaneous field were taken at the solution time corresponding to the end of the simulation ($t = 400(d/u_\infty)$ time units). Figure 6.4 presents a comparison of the instantaneous velocity magnitude at midspan ($y/d = 0$) for the baseline and micro-ramp cases. The flow in the plenum is almost stagnant, but accelerates as it moves into the jet pipe. The contours indicate a flow separation region exists on the downstream side of the jet pipe, which is induced by the flow making the sharp turn from the plenum into the pipe entrance on the downstream side. The low velocity region on the downstream pipe wall induces a high velocity region on the upstream wall as a consequence of conservation of mass, which is known as a "jetting effect" . The separation region and jetting region are sharply delineated by a shear layer, as indicated by the multiple contour lines between the these two regions. At the cross-flow inflow ($x/d \approx -7$) we observe small contour patches created by the turbulent fluctuations entering the domain. Downstream of the jet, the instantaneous velocity field at the midspan appears relatively unaffected by the micro-ramp.

Figure 6.5 shows the instantaneous temperature at midspan for the baseline and micro-ramp flows. On the windward side of the jet, the shear-layer between the jet and cross-flow is seen to become progressively unstable in the streamwise direction, owing to a Kelvin-Helmholtz instability. Downstream, we see that both jets are relatively well attached to the wall due to the shallow inclination of the jet, but the micro-ramp flow appears to have better instantaneous attachment, as evidenced by the apparent lower temperatures near the wall.

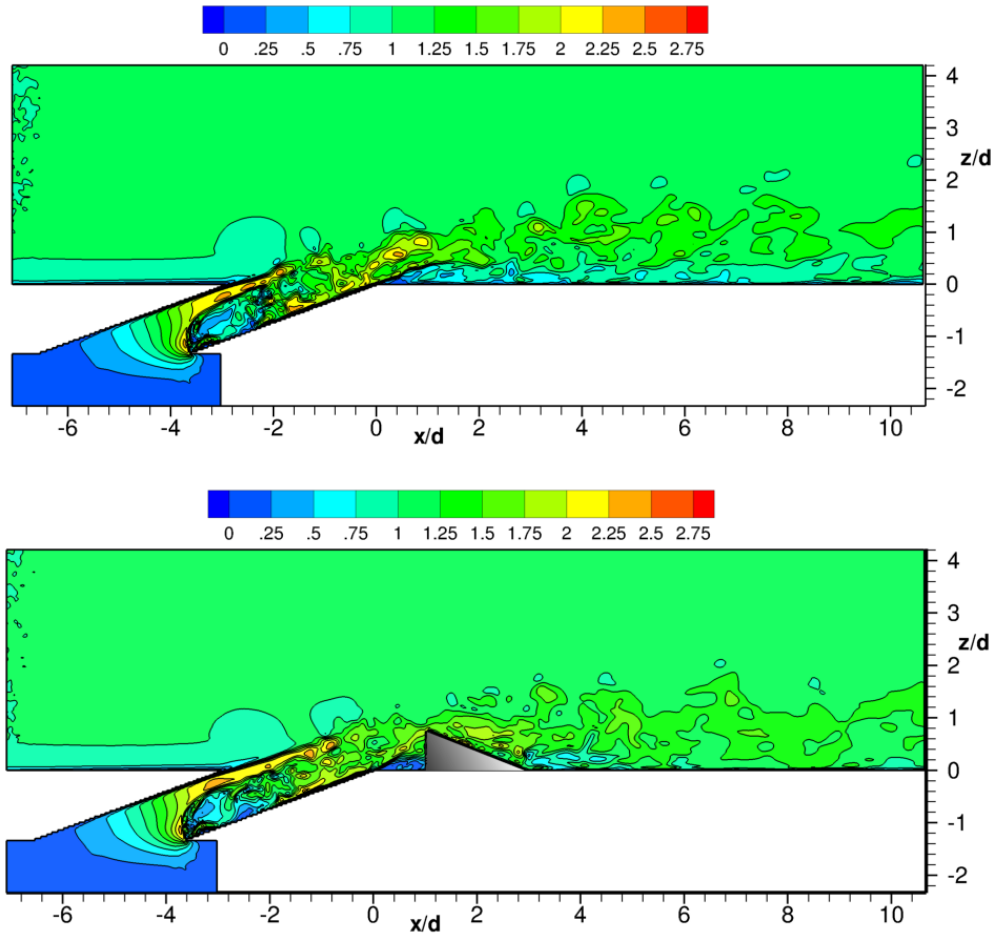


Figure 6.4: Comparison of instantaneous velocity magnitude at midspan ($y/d = 0$).

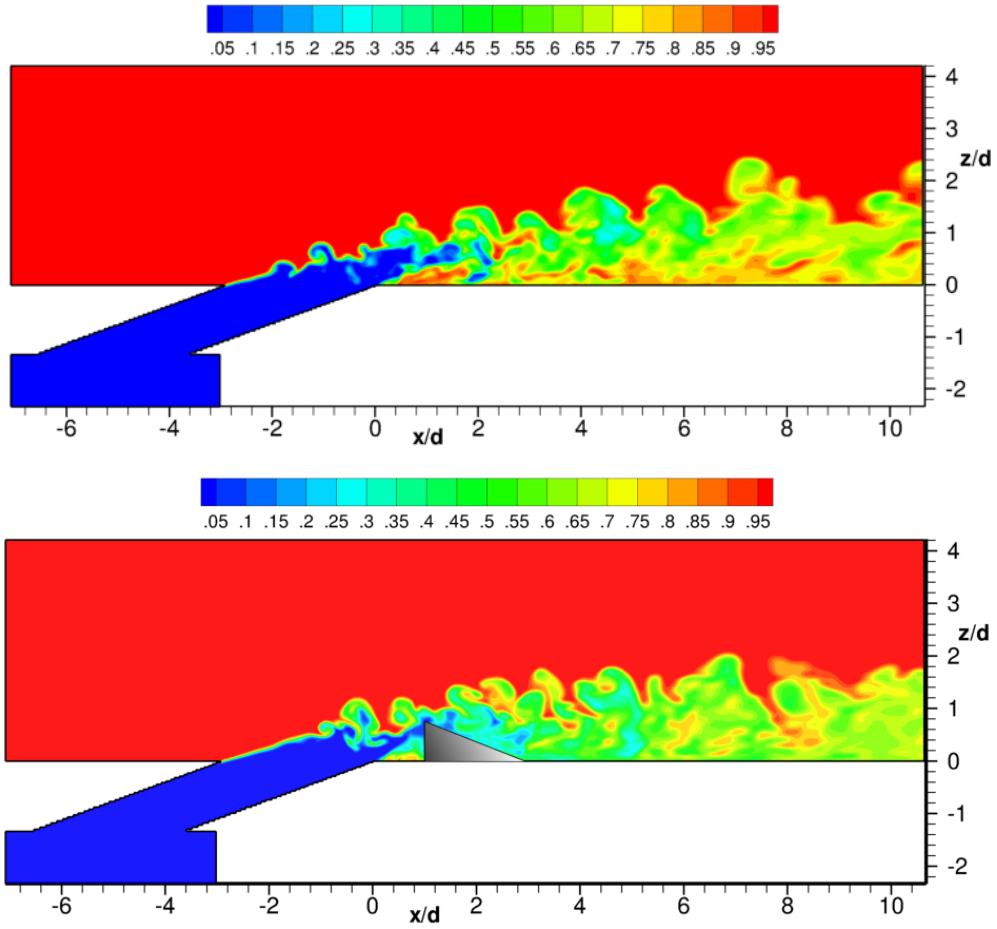


Figure 6.5: Comparison of instantaneous temperature at midspan ($y/d = 0$).

6.6.2 Mean Field

The mean velocity magnitude at midspan is shown in Figure 6.6, along with mean streamlines emanating from the cross-flow inflow and plenum inflow along the midspan. We again see the flow acceleration from the plenum, the separation region on the downstream pipe wall, and the jetting region adjacent to the upstream wall. The streamlines in the plenum show how the fluid moves past the separation region and joins the cross-flow. Downstream, we see similar streamline behavior for both cases, except that there is a slight compression of the lines in the micro-ramp flow. This figure indicates the similarities between the baseline and micro-ramp flows in the spanwise sense only. However, as we will see later, the flows are dramatically different when viewed from a cross-sectional perspective.

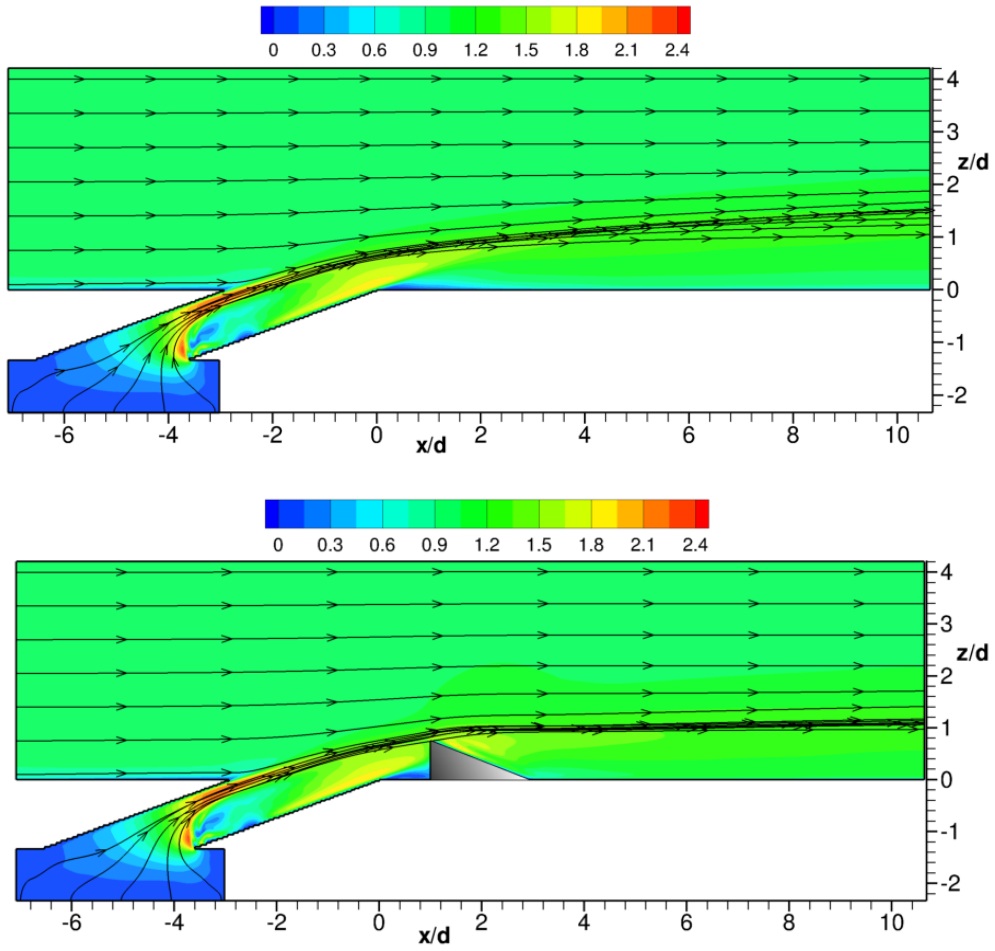
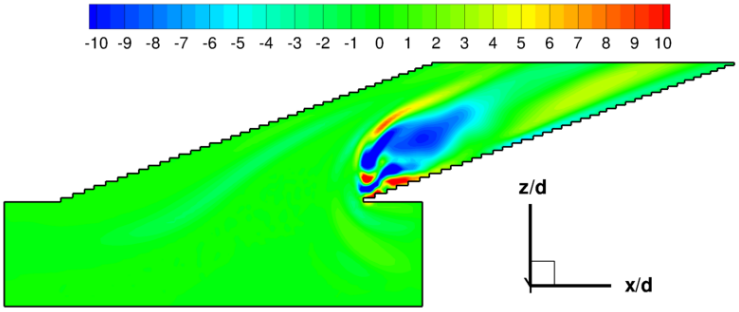


Figure 6.6: Mean velocity magnitude at midspan ($y/d = 0$).

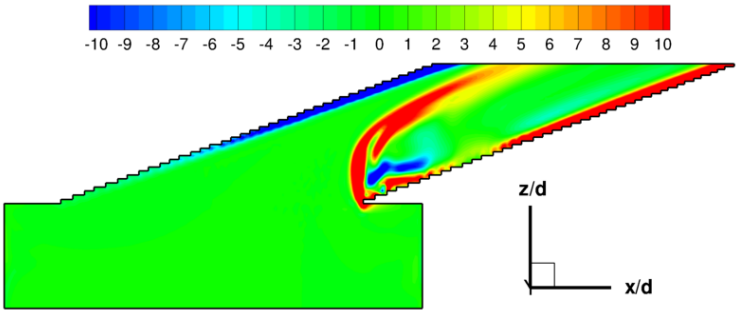
The components and magnitude of mean vorticity in the jet pipe and plenum are shown in Figure 6.7 at midspan. A large region of negative streamwise vorticity is located in the separation region and a strip of positive spanwise vorticity highlights the shear layer between the separation region and jetting region. Mixed patches of positive and negative vertical vorticity are also found in the separation region. The vorticity magnitude shows that the vorticity in the jetting region is relatively low, except for the boundary layer vorticity at the upstream wall. The separation region, however, is dominated by vorticity.

The flow separation and jetting strongly affect the jet exit conditions, as shown for the baseline flow in Figure 6.8, where contours of mean velocity magnitude and mean streamwise vorticity are shown. Note that the cross-flow fluid is moving in the positive x/d direction. The velocity magnitude shows the effect of jetting, where the largest magnitude in velocity is seen near the upstream edge of the exit hole. The lowest magnitudes are located near the center, and the flow recovers a bit near the downstream edge where the velocity has increased relative to the center. The vorticity contours show the counter-rotating vorticity at the wall boundary layers which is being transported into the cross-flow, where larger regions of vorticity are near the leading edge of the jet. This wall vorticity plays a key role in the development of the CRVP of the jet, as will be shown next.

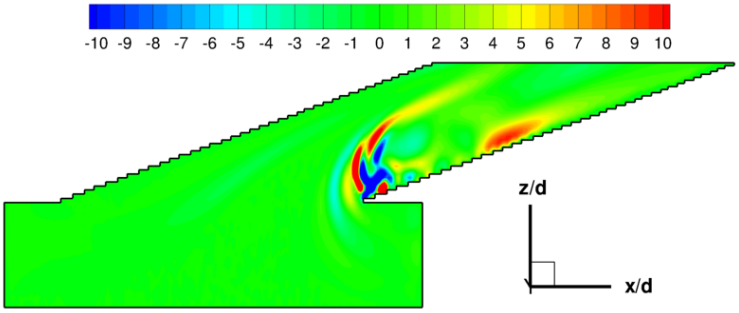
To understand the origin and development of the CRVP in the jet, we examine the jet flow at four streamwise planes indicated in Figure 6.9. The first plane is located over the hole, the second located exactly at the jet trailing edge, and the last two are in the jet near-field. The flow field for these planes is plotted in Figure 6.10, which shows the mean streamwise vorticity and mean in-plane velocity vectors. Note that the jet hole diameter spans from $x/d = -0.5$ to 0.5 and that the cross-flow direction is out of the paper. At $x/d = -1$ (over the hole) we see counter-rotating vortices develop in the flow just above the flat plate near the lateral edges of the jet, where the approximate vertical location of the vortex centers is $z/d = 0.1$. Note there are two sources responsible for the creation of these counter-rotating vortices. One source is the streamwise vorticity from the jet pipe walls



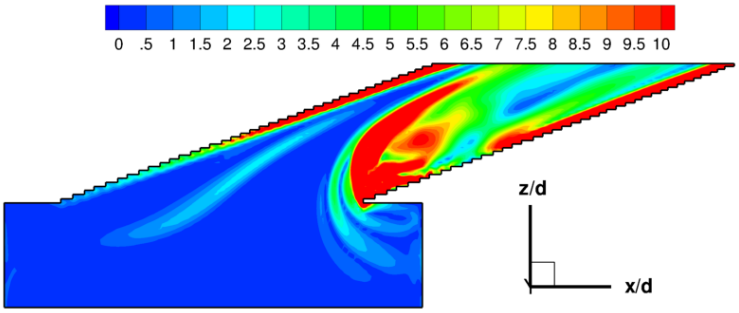
(a) mean streamwise vorticity, $\overline{\omega}_x^*$



(b) mean spanwise vorticity, $\overline{\omega}_y^*$

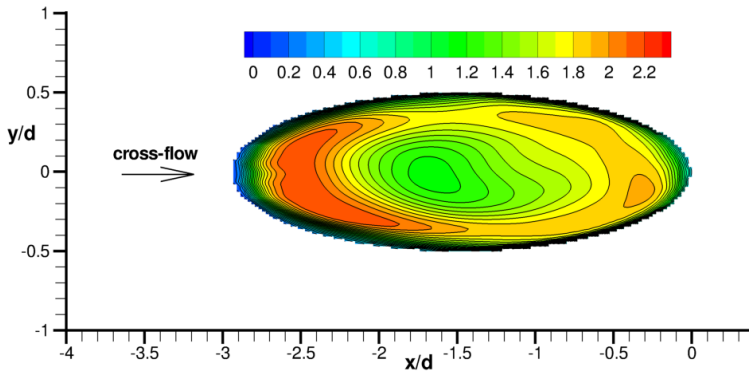


(c) mean vertical vorticity, $\overline{\omega}_z^*$

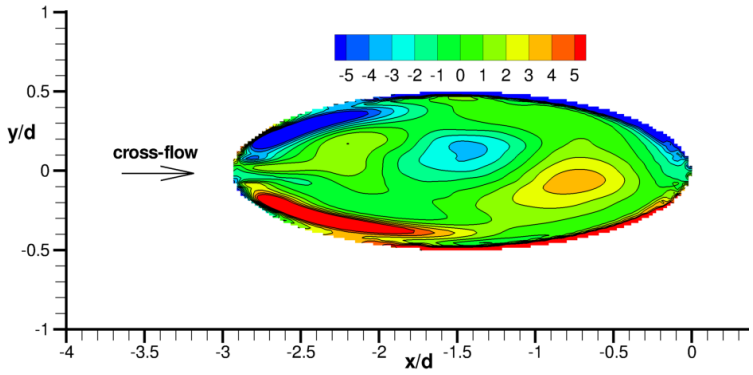


(d) mean vorticity magnitude, $\sqrt{(\overline{\omega}_x^*)^2 + (\overline{\omega}_y^*)^2 + (\overline{\omega}_z^*)^2}$

Figure 6.7: Components and magnitude of mean vorticity in the plenum and jet pipe at midspan ($y/d = 0$).



(a)



(b)

Figure 6.8: Mean velocity magnitude (a) and mean streamwise vorticity (b) at jet inflow ($z/d = 0$) for the baseline flow.

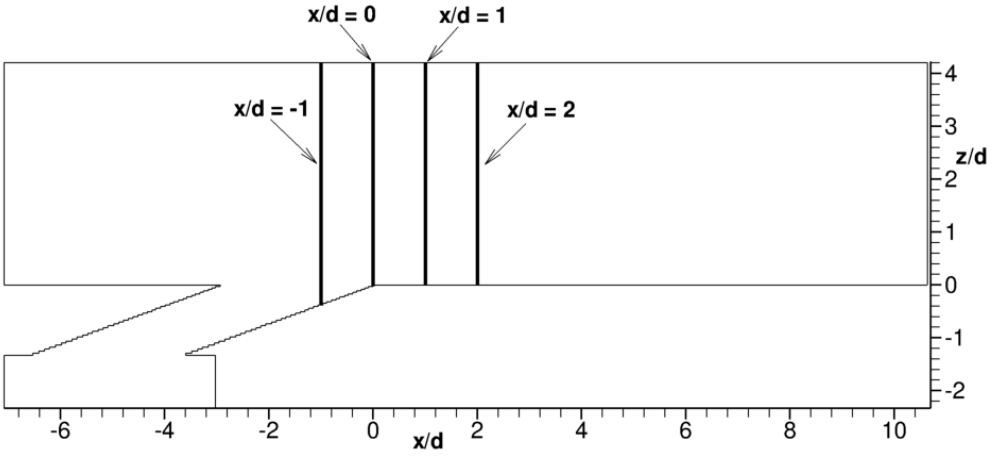
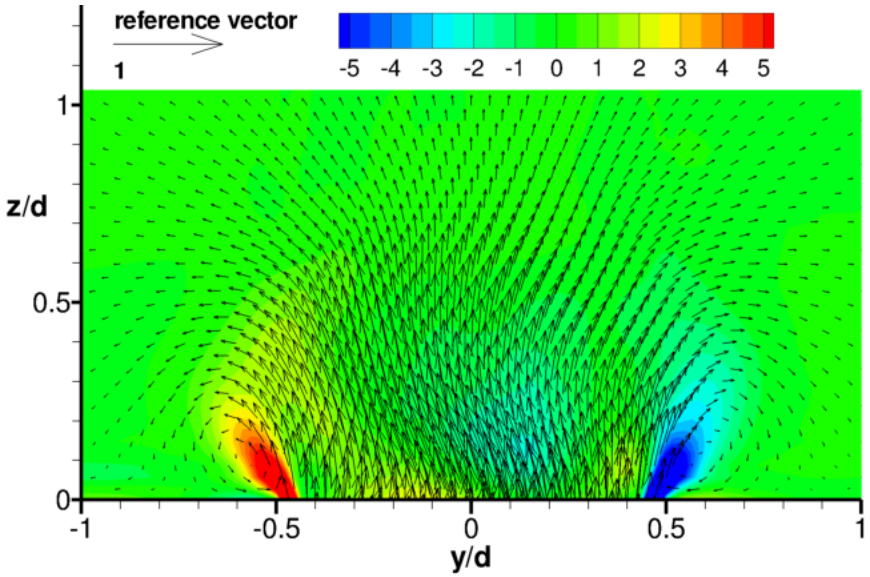


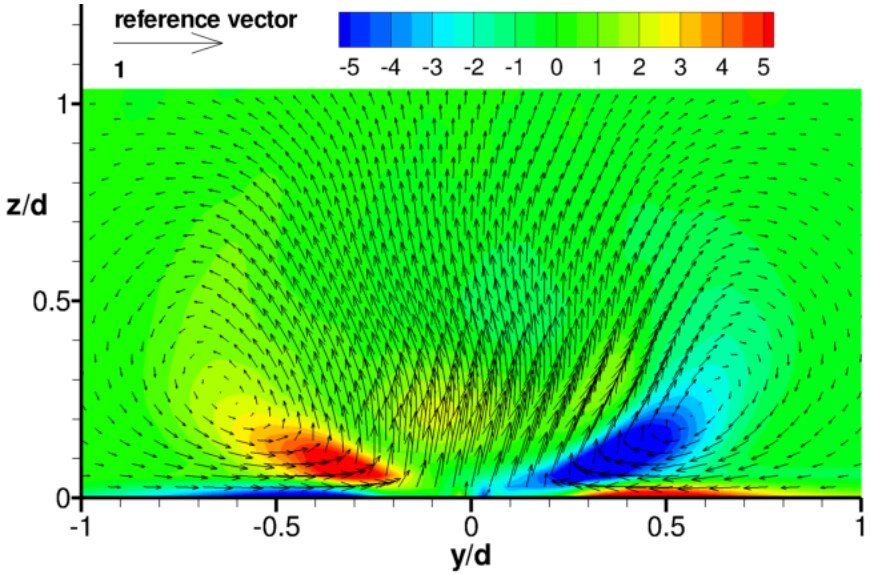
Figure 6.9: Location of survey planes in near-field of jet.

entering into the cross-flow and the other is the shearing of the cross-flow by the jet which generates streamwise vorticity near the lateral edges of the jet exit. Farther downstream at $x/d = 0$ (jet trailing edge), we see that the centers of the vortices have moved away from the wall to approximately $z/d = 0.3$. At $x/d = 1$, the vortices have moved closer together in the spanwise direction such that they are now in contact, where the dividing plane is at midspan. Finally, at $x/d = 2$, the CRVP has clearly developed into the dominant mean flow structure in the plane and the centers are now at approximately $z/d = 0.5$.

The next issues to address are what happens in the jet far field and what happens to the CRVP structure if a micro-ramp is added. These issues are addressed by again plotting along streamwise planes, but this time at the locations indicated in Figure 6.11. At each of these planes, the mean streamwise vorticity and mean streamlines are plotted as shown in Figure 6.12. Note that the cross-flow is directed out of the paper. The left column is for the baseline flow, and the right column is for the flow with a micro-ramp. For the micro-ramp case, the triangular projection of the entire micro-ramp cross-section is indicated in black lines, and for the plane that intersects the micro-ramp ($x/d = 2$) the cross-section of the micro-ramp is shown in gray. The first plane is at $x/d = 0.5$, which is located halfway between the jet trailing-edge and the micro-ramp leading edge. Here we see little difference between the two cases, which confirms the expected result that the micro-ramp has a negligible influence on

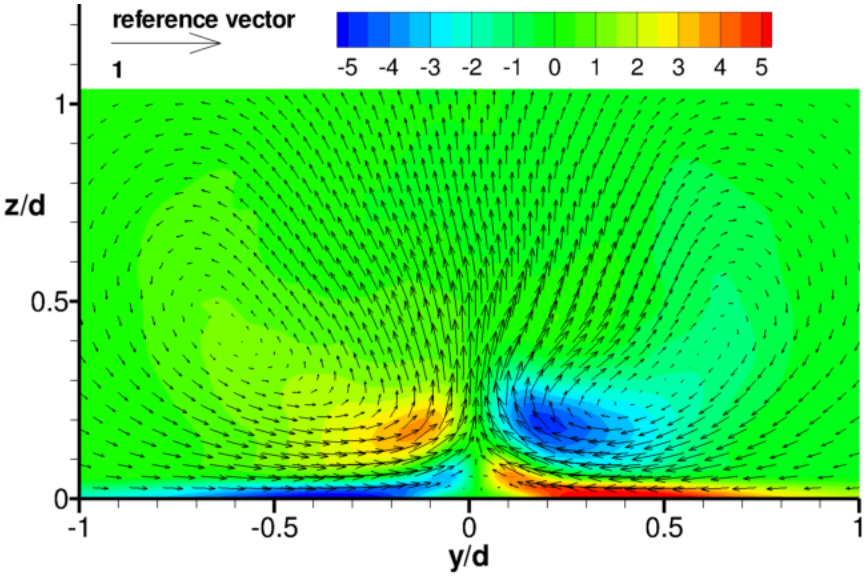


(a) $x/d = -1$

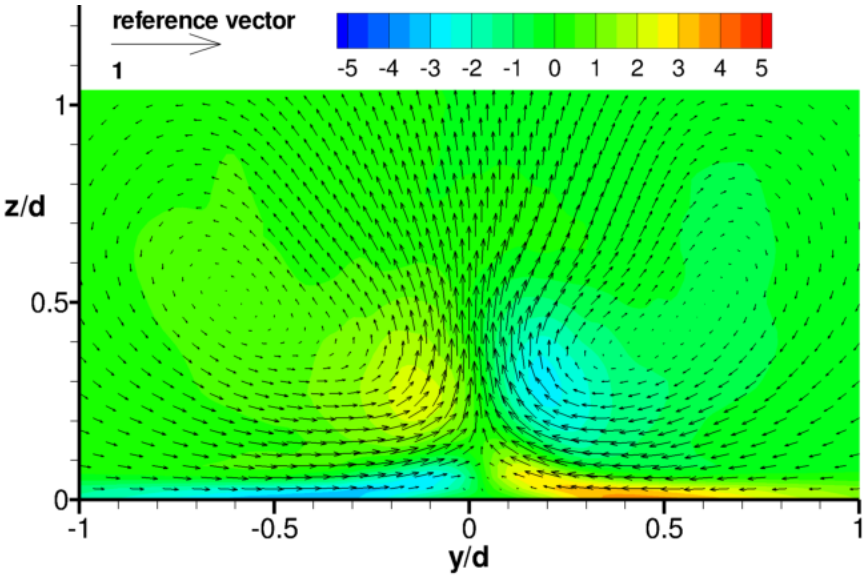


(b) $x/d = 0$

Caption on next page.



(c) $x/d = 1$



(d) $x/d = 2$

Figure 6.10: Streamwise evolution of the CRVP in the jet. Mean streamwise vorticity and mean velocity vectors are shown at streamwise slices in the jet near-field.

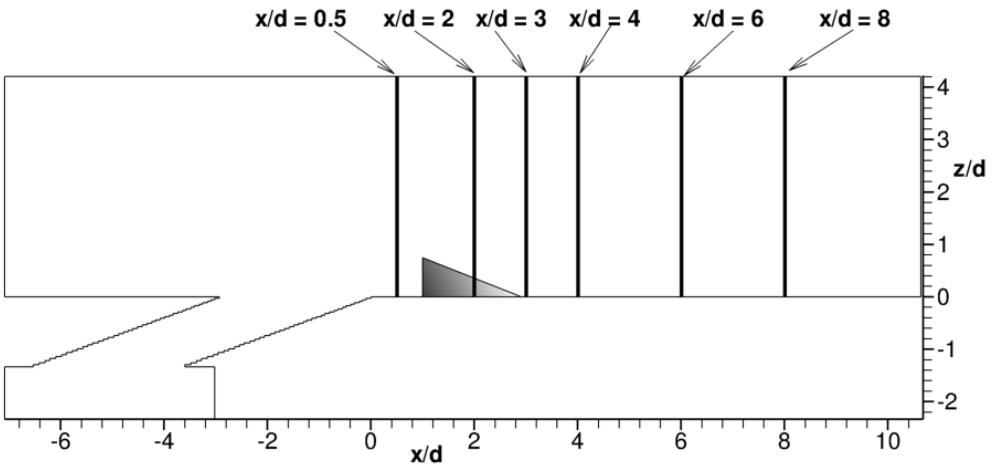
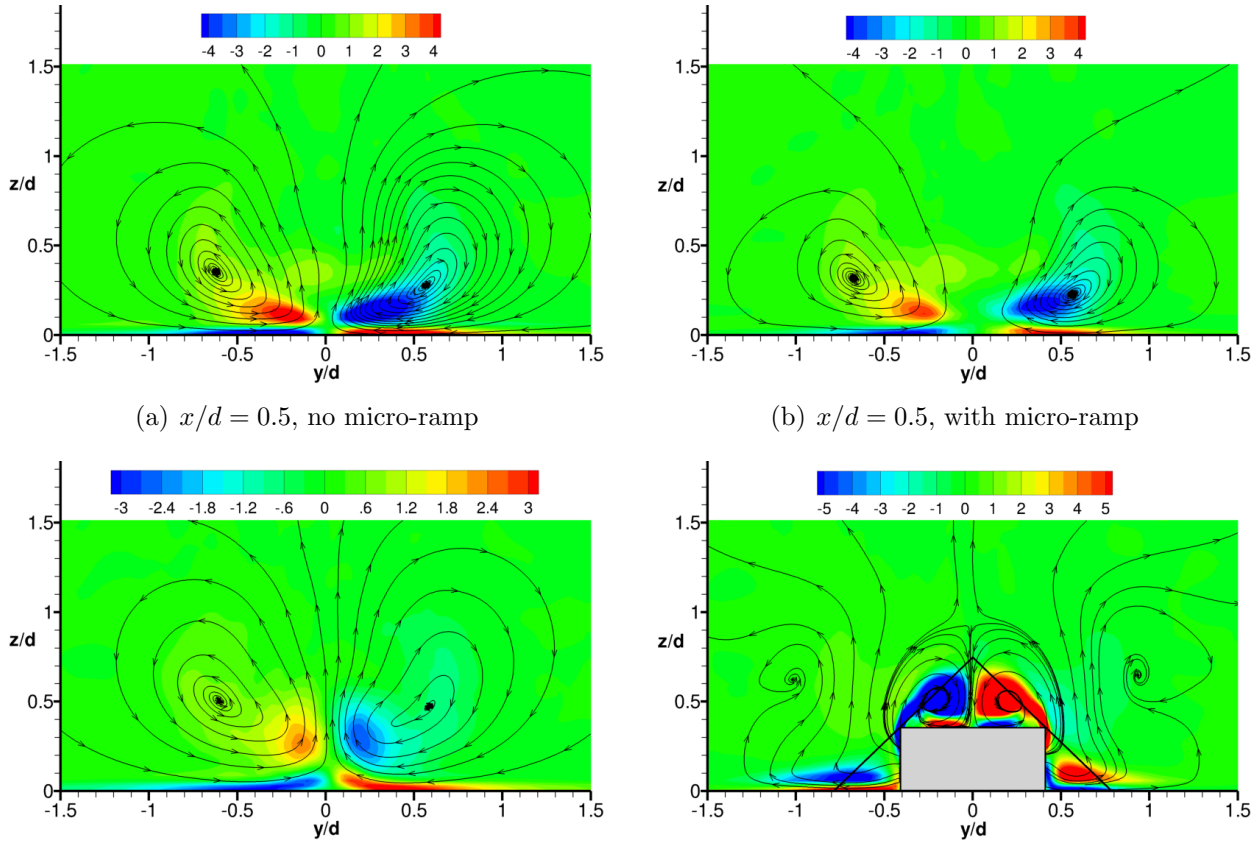


Figure 6.11: Location of streamwise survey planes for vorticity and temperature.

the upstream flow.

At $x/d = 2$, the plane is passing through the micro-ramp and we see a dramatic difference between the two cases. The micro-ramp generates counter-rotating vortices of opposite sense to the CRVP; at $x/d = 2$, these vortices are located on top of the micro-ramp surface. The vorticity in the CRVP was decreased relative to the baseline case, owing to vorticity cancellation from the micro-ramp vortices. The streamlines show that the jet vortices in the CRVP have been pushed out laterally and moved up vertically due to the micro-ramp. At $x/d = 3$ and $x/d = 4$, the micro-ramp vortices have moved downstream and are now at the flat plate wall where they create a downwash effect at the midspan. Note that this is exactly the opposite of the baseline flow, where the CRVP of the jet created an upwash effect at midspan. The CRVP persists at these planes but is weak compared with the micro-ramp vortices. Farther downstream at $x/d = 6$ and $x/d = 8$, the micro-ramp vortices have weakened but still dominate the CRVP and produce a downwash effect at the midspan. In summary, it appears that the micro-ramp is capable of generating a strong vorticity field with a beneficial downwash effect that can help transport jet coolant to the wall, and through vorticity cancellation, the micro-ramp vortices weaken the jet CRVP.

We now examine the mean temperature field. The mean temperature at midspan is shown in Figure 6.13. The vertical penetration of the jet coolant into the cross-flow is



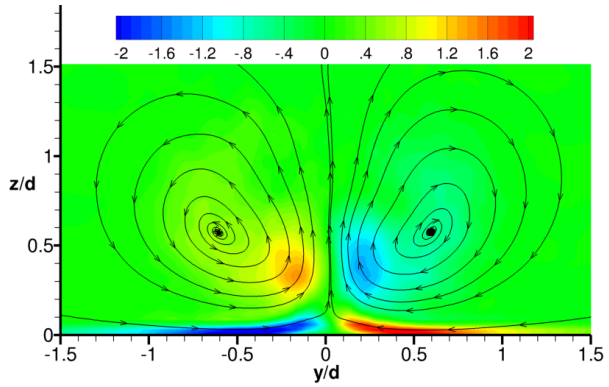
(a) $x/d = 0.5$, no micro-ramp

(b) $x/d = 0.5$, with micro-ramp

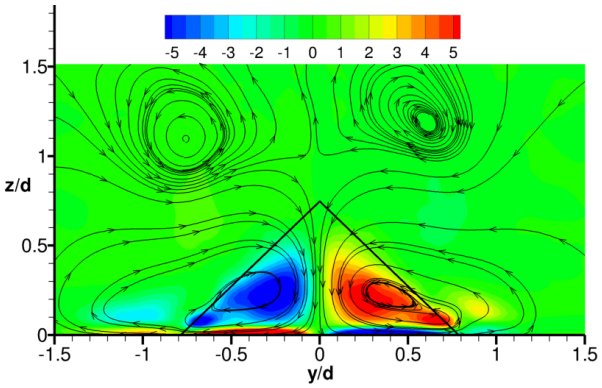
(c) $x/d = 2$, no micro-ramp

(d) $x/d = 2$, with micro-ramp

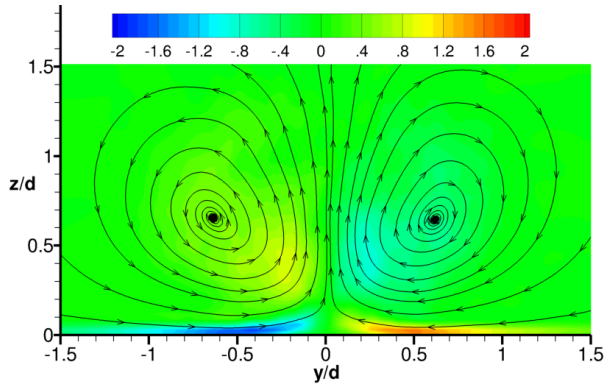
Caption on page 173.



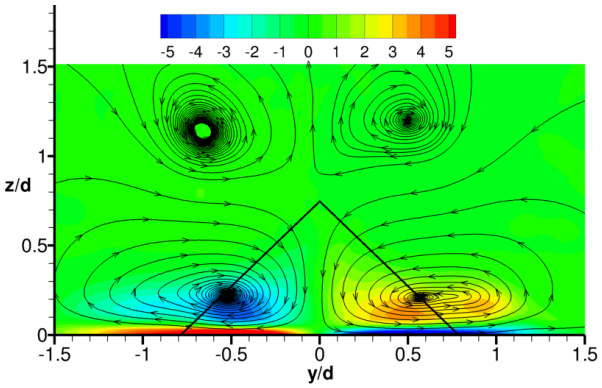
(e) $x/d = 3$, no micro-ramp



(f) $x/d = 3$, with micro-ramp

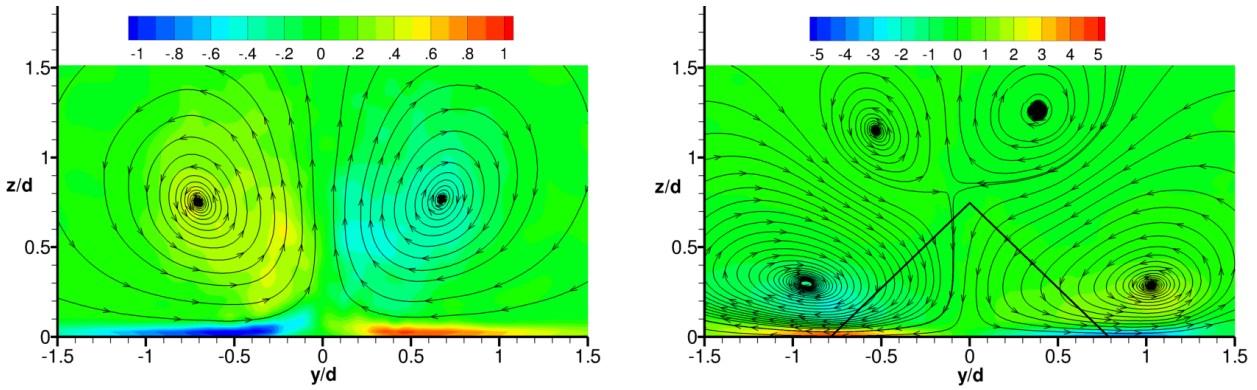


(g) $x/d = 4$, no micro-ramp



(h) $x/d = 4$, with micro-ramp

Caption on page 173.



(i) $x/d = 6$, no micro-ramp

(j) $x/d = 6$, with micro-ramp

(k) $x/d = 8$, no micro-ramp

(l) $x/d = 8$, with micro-ramp

Figure 6.12: Mean streamwise vorticity and streamlines at streamwise planes.

relatively the same for both cases. In addition, both jets appear well attached at the wall. The primary (and most important) difference is the lower mean temperature near the wall using the micro-ramp. The cooling effectiveness of the micro-ramp is better displayed by viewing the mean temperature distribution along the entire wall, as done in Figure 6.14. The baseline jet only cools a narrow strip near the centerline of the wall, wheres the micro-ramp cools a much wider area and drives the minimum wall temperature lower than that achieved by the baseline flow.

The streamwise evolution of the mean temperature is shown in Figure 6.15 along the planes indicated in Figure 6.11. The results are presented in the same format as before, where the cross-flow is directed out of the paper and the micro-ramp's full triangular projection is indicated with black lines and intersection with the plane is indicated in gray. The streamwise evolution of the baseline jet indicates that it remains attached, but only for a narrow region at the wall. One explanation for this is that the CRVP rotates such that it entrains hot cross-flow and moves it toward the wall, causing a contraction of the cooled area. There is not much lateral spreading of the baseline jet, but there is a noticeable increase in vertical jet penetration. This latter feature is due to the mutual induction from the CRVP, which creates a tendency for the jet flow to lift vertically. Also note the upward bulging of the temperature contours near the midspan of the wall at $x/d = 2, 3, 4$ which is due to the upwash effect of the CRVP. This effect is detrimental to cooling effectiveness since it moves coolant away from the wall.

Now we will compare the baseline and micro-ramp temperature fields. At $x/d = 0.5$ the baseline and micro-ramp temperature fields are nearly the same since there is no upstream influence from the micro-ramp. At $x/d = 2$ (plane through the micro-ramp) we begin to see some lateral expansion of the coolant near the wall for the micro-ramp case. This lateral expansion of the coolant increases at $x/d = 3$ and $x/d = 4$, which is caused by the transport of coolant by the counter-rotating vortices from the micro-ramp toward the wall and laterally along the wall. Farther downstream at $x/d = 6$ and $x/d = 8$ the lateral spreading continues,

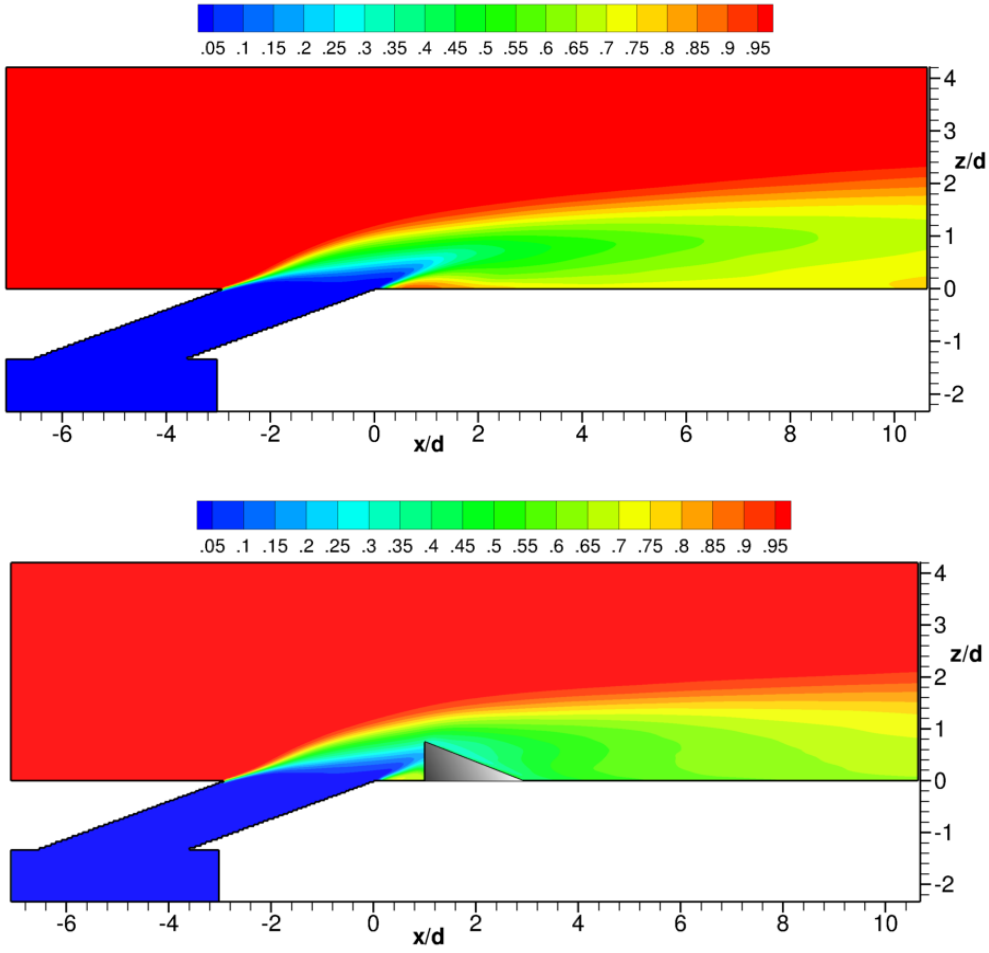


Figure 6.13: Mean temperature at midspan ($y/d = 0$).

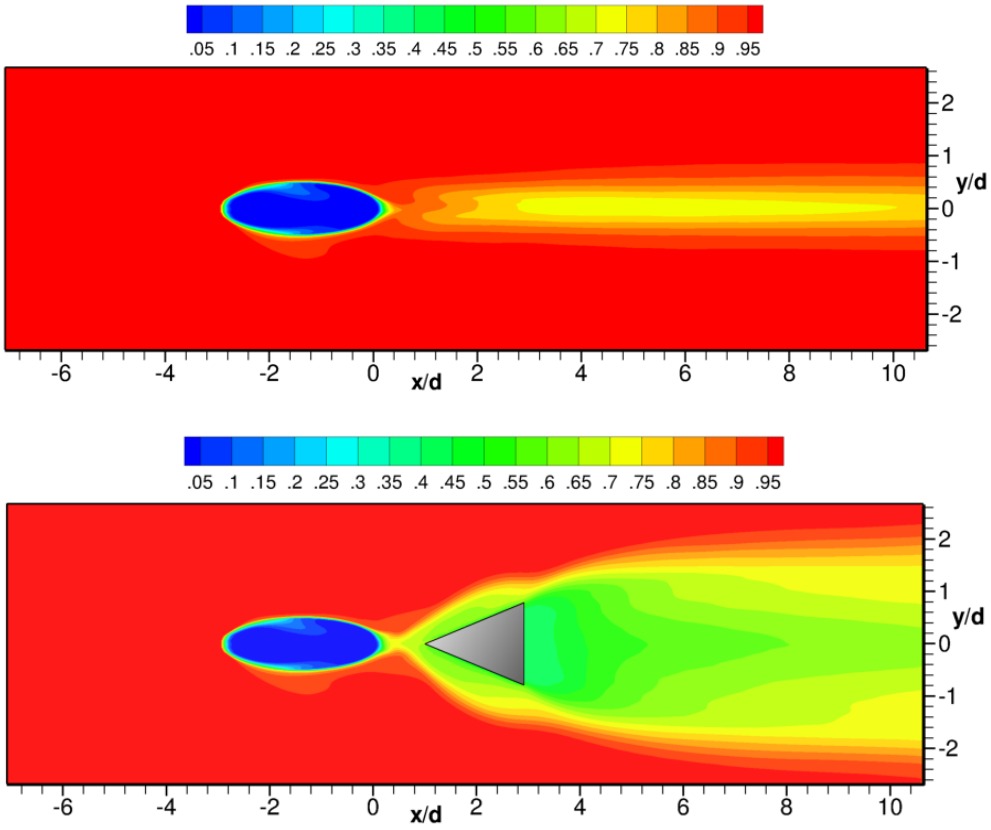
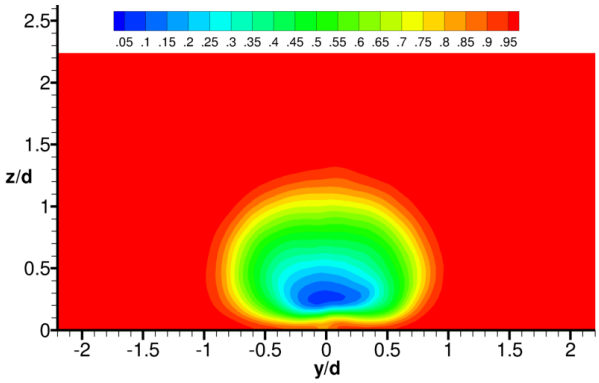
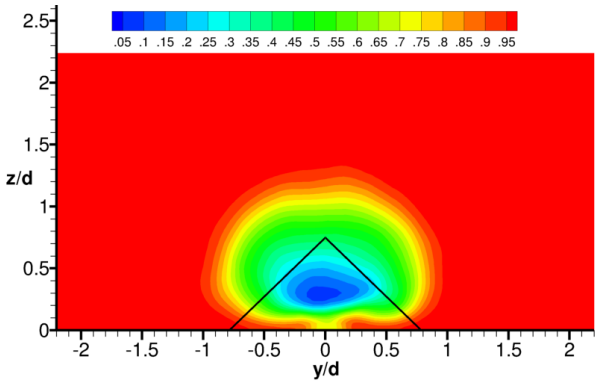


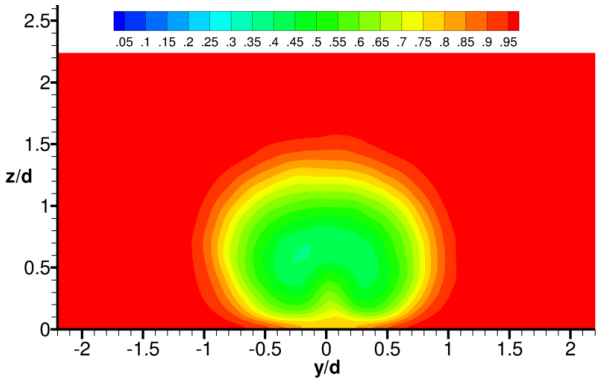
Figure 6.14: Mean temperature along wall of domain ($z/d = 0$).



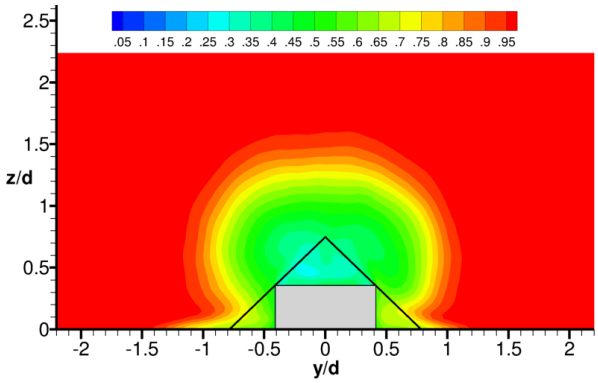
(a) $x/d = 0.5$, no micro-ramp



(b) $x/d = 0.5$, with micro-ramp

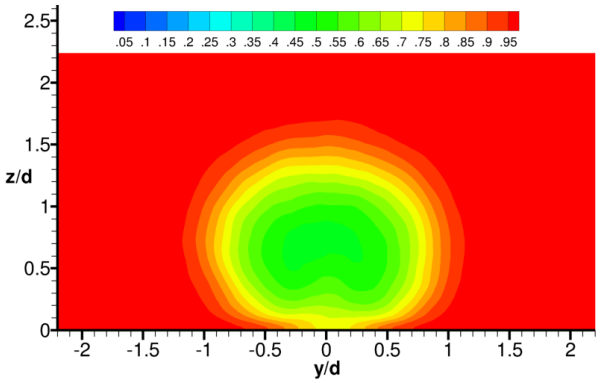


(c) $x/d = 2$, no micro-ramp

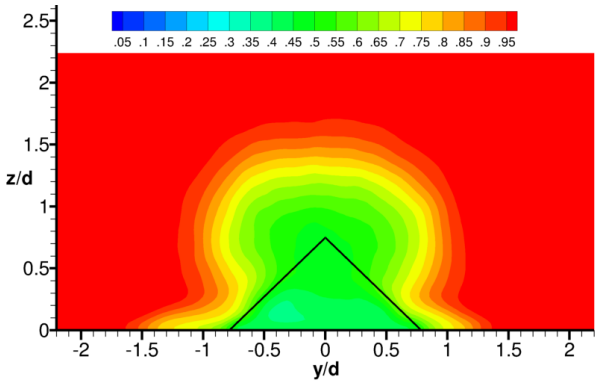


(d) $x/d = 2$, with micro-ramp

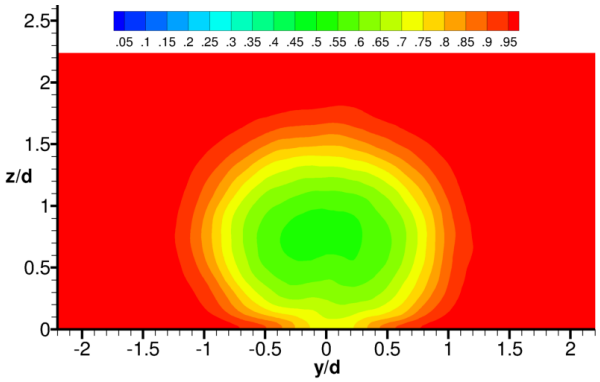
Caption on page 179.



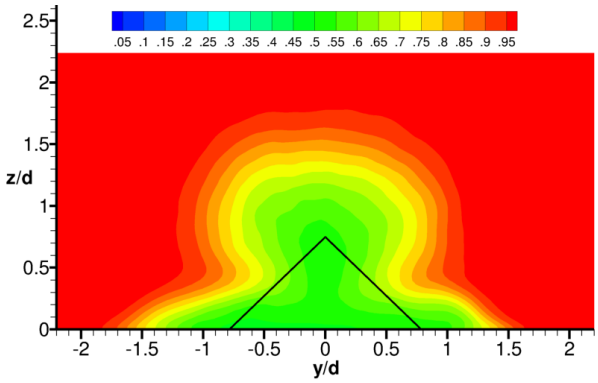
(e) $x/d = 3$, no micro-ramp



(f) $x/d = 3$, with micro-ramp

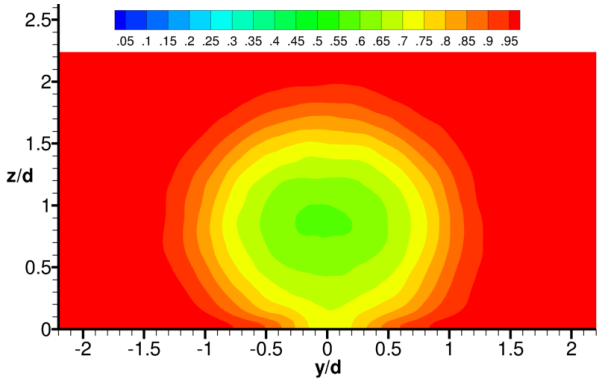


(g) $x/d = 4$, no micro-ramp

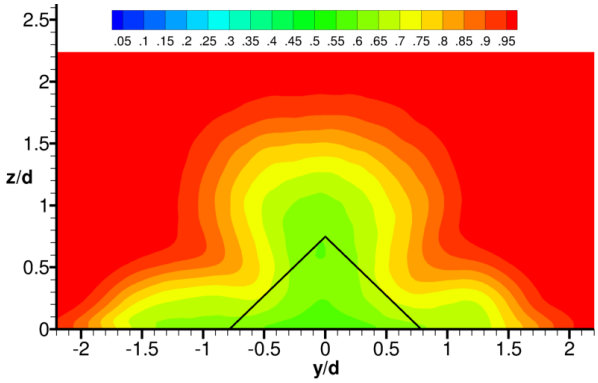


(h) $x/d = 4$, with micro-ramp

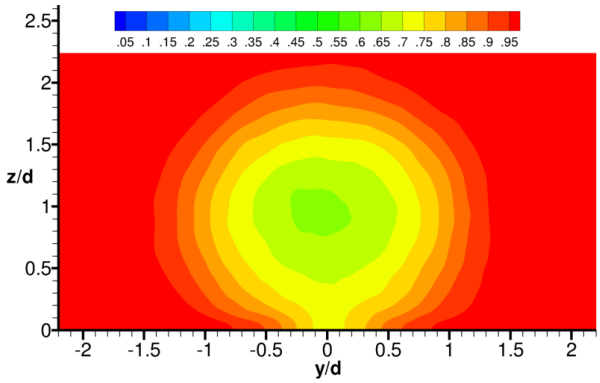
Caption on page 179.



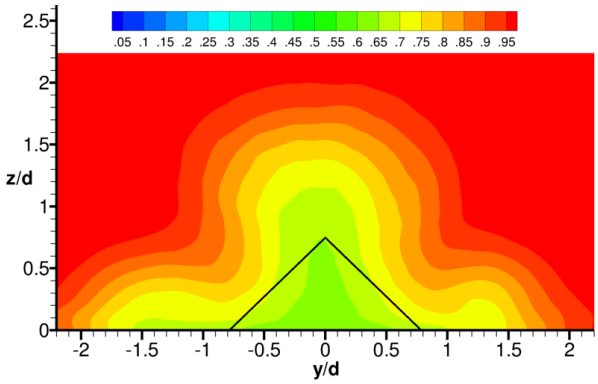
(i) $x/d = 6$, no micro-ramp



(j) $x/d = 6$, with micro-ramp



(k) $x/d = 8$, no micro-ramp



(l) $x/d = 8$, with micro-ramp

Figure 6.15: Mean temperature at streamwise planes.

resulting in more surface coverage by the coolant. For the micro-ramp case, the weakened CRVP from the jet appears to have little influence on the temperature distribution. Thus, the dominant flow feature is the vortex pair from the micro-ramp and is found to have a favorable effect on the cooling effectiveness that persists downstream into the jet far-field.

6.6.3 Film-Cooling Effectiveness

In the previous chapter, two important metrics for film-cooling flows were introduced: the film-cooling effectiveness and span-averaged film-cooling effectiveness. Respectively, these are computed as

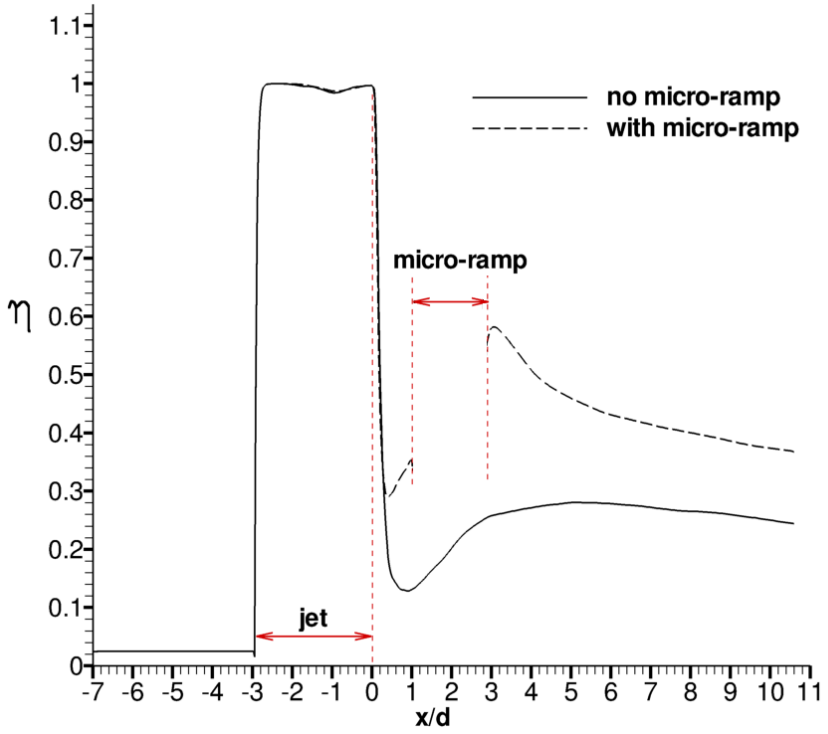
$$\eta = \frac{(T_\infty - \overline{T}_w)}{(T_\infty - T_j)} = 1 - \overline{T}^*_w \tag{6.37}$$

and

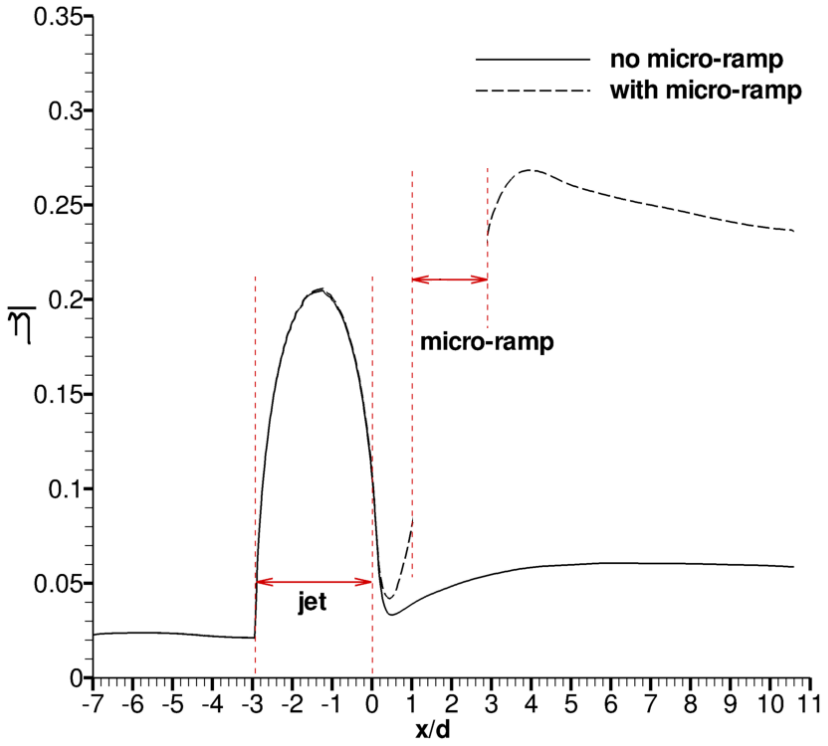
$$\overline{\eta}(x) = \frac{1}{L_y} \int_{-L_y/2}^{L_y/2} \eta(x, y)\ dy \tag{6.38}$$

The film-cooling effectiveness evaluated at the centerline of the domain and span-averaged film-cooling effectiveness are plotted in Figure 6.16 for the baseline and micro-ramp cases. The effectiveness was evaluated at the flat plate surface ($z/d = 0$) only. Since the micro-ramp surface is elevated above this, the region where the micro-ramp is located was omitted from the curves in order to be consistent.

For the centerline effectiveness, a value of almost zero is observed until the jet leading edge, where a step-change occurs due to injection of coolant, and the effectiveness rises to unity. This is observed for either the baseline or micro-ramp case, since this all occurs upstream of the micro-ramp. At the jet trailing edge, the effectiveness drops sharply for both cases owing to jet separation immediately downstream of the jet hole. This was more severe for the baseline case, but it soon recovers as the jet attaches to the wall and then drops again downstream due to mixing with the hot cross-flow and jet lift-off. For the micro-ramp we see improved effectiveness compared to the baseline, which peaks just downstream of the



(a) film-cooling effectiveness along centerline of domain



(b) span-averaged film-cooling effectiveness

Figure 6.16: Film-cooling effectiveness comparison between baseline flow and flow with micro-ramp.

micro-ramp and decays thereafter. This decay is due to the weakening of the micro-ramp vortices, which cannot sustain the transport of coolant to the wall. For the span-averaged effectiveness, we see a similar trend. Note that in the jet we do not see a unity effectiveness since the integration across the domain includes high wall temperatures on the lateral sides of the jet. Again, we see downstream improvement by using a micro-ramp. The factor of improvement is larger for the span-averaged values compared with the centerline values because the baseline case was dominated by high temperatures in the spanwise direction and the micro-ramp flow had a wider region of lower wall temperatures, hence integration across the span amplifies the differences.

6.6.4 Comparison with Experiment

Figure 6.17 shows a comparison of the predicted mean boundary-layer profile for the cross-flow versus experiment at a streamwise location of $x/d = -2.95$. Away from the wall the agreement is excellent, but near the wall we see that the LES profile has laminarized. The reason is that turbulent fluctuations were damped in the boundary layer, reducing the instantaneous velocity and therefore the mean. This was due in part to using a downstream r.m.s. profile to construct the upstream inflow boundary condition, since no experimental data were known far upstream.

The plenum boundary conditions are checked by verifying that the mean blowing ratio matches the experimental value. This was done by first calculating the volume flow rate through four cross-sectional survey planes in the jet pipe (Figure 6.18) based on the mean flow solution. Dividing this by the discrete cross-sectional area of the jet pipe yielded the bulk jet velocity $u_j^* = u_j/u_\infty$, which is the same as the blowing ratio. The four blowing ratios were averaged to get a single representative value of the blowing ratio, which was found to be approximately 1.3913. The absolute relative error between this predicted blowing ratio and the experimental blowing ratio of $\sqrt{2}$ is approximately 1.6 percent, which is reasonable agreement.

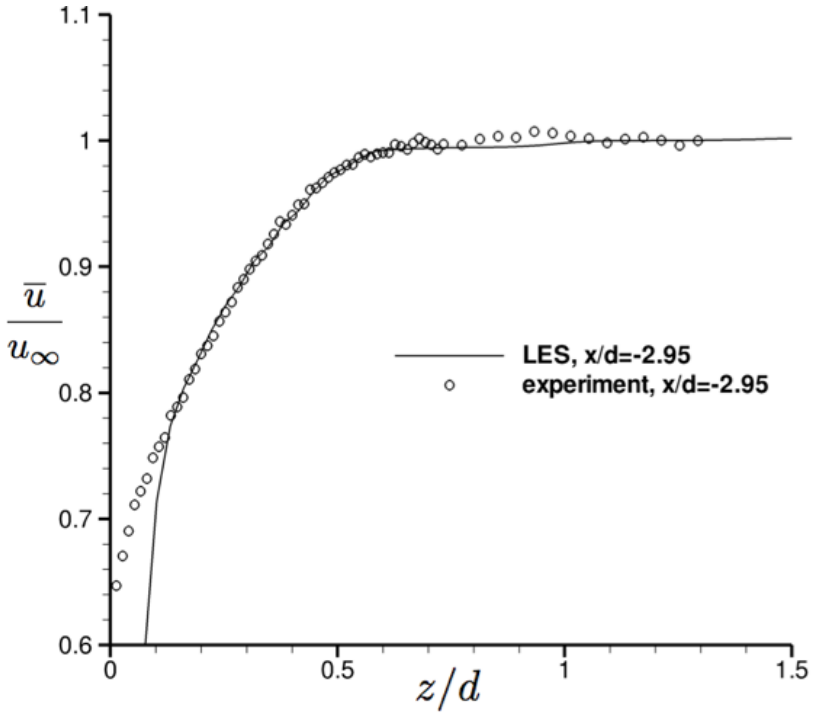


Figure 6.17: Comparison of predicted mean boundary-layer profile versus experiment at centerline of domain ($y/d = 0$).

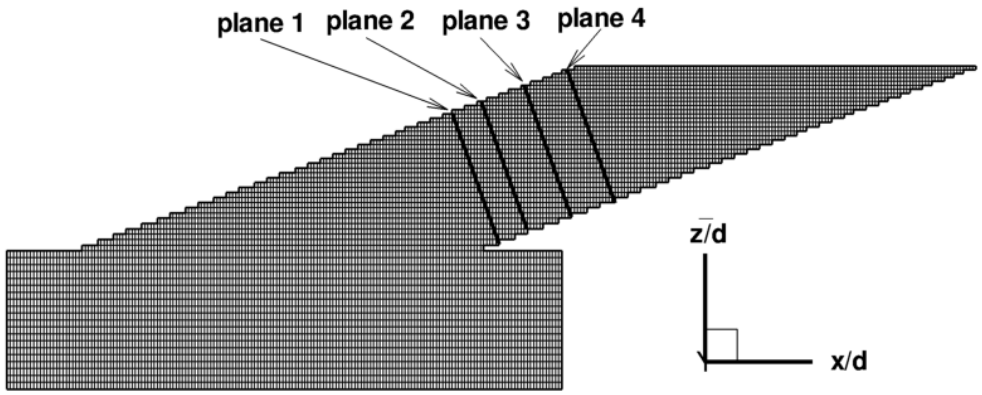


Figure 6.18: Location of cross-sectional survey planes for volume flow rate through jet pipe.

Figure 6.19 compares the predicted vortex core trajectories with the experiment, where the trajectories are plotted two-dimensionally in the streamwise direction (x/d) and the spanwise direction (y/d); thus the trajectories are viewed from above looking down at the flat plate wall. For each case two lines are plotted, where one line is for the vortex core on the positive side of the domain (solid line) and the other for the vortex core on the negative side (broken line).

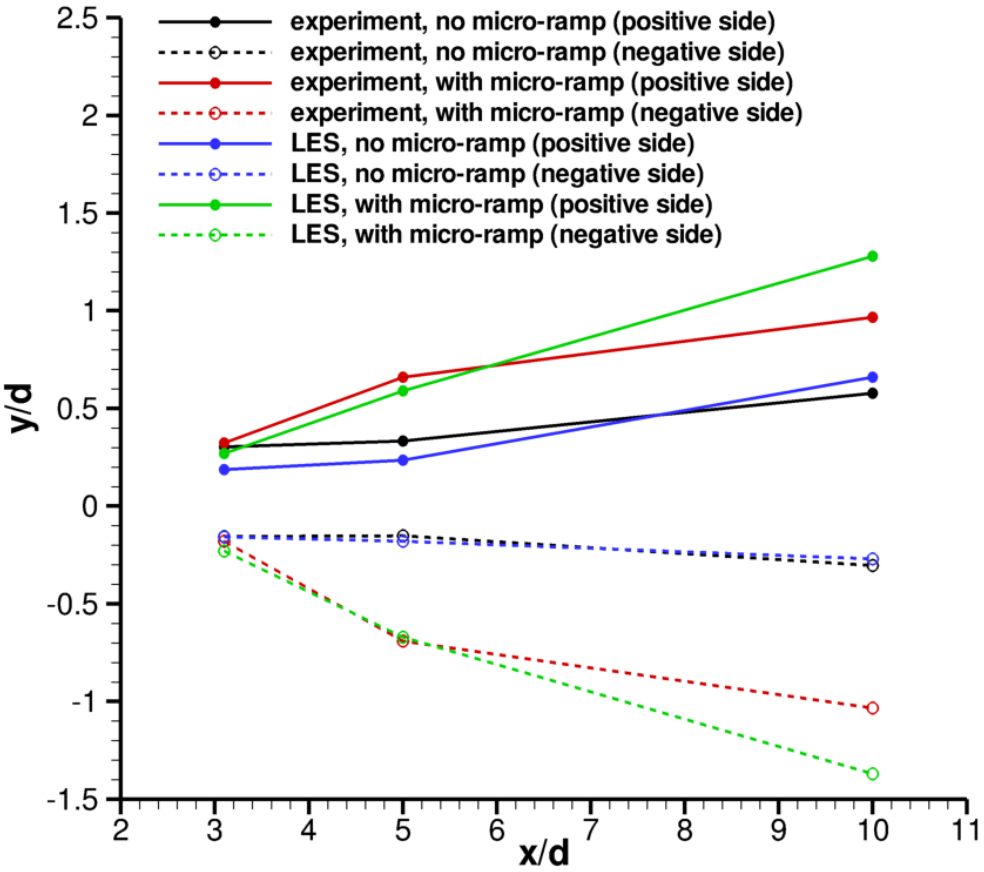


Figure 6.19: Comparison of vortex core trajectories.

In this plot, each symbol represents a single value, and only three values are plotted per line. The reason is that only three streamwise locations were reported in the experiment for the domain length considered in the LES. While the data are coarse, this still gives an interesting comparison. The spanwise trajectory locations (y/d) from the LES were selected using the location of the maximum and minimum mean streamwise vorticity, which

is how they were determined in the experiment. For the case with no micro-ramp, the predicted trajectory matches reasonably well versus experiment, especially for the vortex on the negative side. With the micro-ramp, we see that in the near-field (up to $x/d = 5$) the agreement is good, but in the far-field at $x/d = 10$, there is disagreement, where the simulation produced an over-prediction in the lateral spreading of the jet.

Table 6.2 compares the maximum positive mean streamwise vorticity, $\max\{\overline{\omega_x}^*\}$, from the LES predictions versus experiment at three streamwise locations. For the case with no micro-ramp the agreement is very reasonable. However, for the case with the micro-ramp the agreement is mixed. At $x/d = 3.1$ the LES over-predicted the experimental measurements of vorticity by approximately a factor of two. The agreement improves at $x/d = 5$ but worsens further downstream at $x/d = 10$, where the LES over-predicted the vorticity by a factor of approximately 1.4. Table 6.3 shows a comparison of maximum r.m.s. streamwise velocity, $\max\{u^*_{rms}\}$, for the LES and experiment. For the case with no micro-ramp the values obtained from LES show an over-prediction of approximately a factor of two. For the case with the micro-ramp the values show an over-prediction that is less severe, where the worst over-prediction occurs at $x/d = 10$ by a factor of approximately 1.8.

Table 6.4 shows a comparison of the vertical location (z/d) of maximum mean streamwise velocity $\max\{\overline{u}^*\}$ for the LES and experiment, which is an approximate way to measure the jet trajectory in the vertical direction (or "jet penetration"). For the case with no micro-ramp, the LES over-predicts the trajectory in the near-field at $x/d = 3.1$, agrees very well at $x/d = 5$, but then under-predicts in the far-field at $x/d = 10$; this indicates the overall slope of the jet is shallower than was observed experimentally. With the micro-ramp, we see better agreement in the near-field at $x/d = 3.1$ but over-prediction farther downstream; thus the LES predicts that the jet flow experiences more lift-off than seen in the experiment. Overall, the simulation and experimental data in Table 6.4 both indicate the addition of the micro-ramp causes the jet trajectory to move closer to the wall, which is the desired effect for improving film-cooling at the wall. It is believed that the disagreements noted above are

mostly due to the uncertainties in the construction of the inflow boundary conditions.

Table 6.2: Comparison of LES versus experiment for maximum mean streamwise vorticity, $\max\{\overline{\omega_x}^*\}$.

	no micro-ramp		with micro-ramp	
x/d	LES	experiment	LES	experiment
3.1	1.376	1.219	5.30	2.764
5	0.678	0.652	2.23	2.269
10	0.291	0.217	1.01	0.712

Table 6.3: Comparison of LES versus experiment for maximum r.m.s. streamwise velocity, $\max\{u^*_{rms}\}$.

	no micro-ramp		with micro-ramp	
x/d	LES	experiment	LES	experiment
3.1	0.274	0.153	0.329	0.217
5	0.230	0.114	0.199	0.156
10	0.206	0.086	0.180	0.099

Table 6.4: Comparison of LES versus experiment for vertical location (z/d) of maximum mean streamwise velocity, $\max\{\overline{u}^*\}$.

	no micro-ramp		with micro-ramp	
x/d	z/d (LES)	z/d (experiment)	z/d (LES)	z/d (experiment)
3.1	0.925	0.619	0.77	0.711
5	1.04	0.999	0.85	0.631
10	1.272	1.437	0.93	0.629

6.7 Conclusions and Recommendations

In this chapter, Large Eddy Simulations were presented to study a film-cooling configuration that included the coolant delivery pipe and plenum. This configuration was designed to model the experiment performed by Zaman et al. [23]. The purpose of this study was to compare how a micro-ramp vortex generator placed downstream of the jet alters the flow

field compared to a baseline case with no micro-ramp, where film-cooling effectiveness was the primary focus. Two simulations were performed: a baseline flow with no micro-ramp and a flow with a micro-ramp included one diameter downstream of the jet trailing edge. Comparisons were made with experimental measurements, and results were presented using the instantaneous and time-averaged flow fields to elucidate the physics. The following text summarizes the conclusions of this chapter:

1. The LES results were compared to measurements from the experiment of Zaman et al. [23]. The predicted vortex core trajectories in the spanwise direction matched reasonably well versus experiment for the baseline case. The agreement was also good for the micro-ramp case in the near-field, but in the far-field the simulations over-predicted the lateral spreading of the jet. A comparison of the maximum positive mean streamwise vorticity showed reasonable agreement for the baseline case, but the micro-ramp case showed some over-prediction. A comparison of jet penetration for the baseline case showed that the LES predicted an overall shallower slope of the jet than was observed experimentally; for the micro-ramp case the LES predicted more jet lift-off than seen in the experiment. Both simulation and experiment indicate that the addition of a micro-ramp causes the jet trajectory to move closer to the wall.

2. An analysis of the flow in the plenum and jet pipe showed a separation region developed on the downstream pipe wall owing to the sharp turn at the plenum exit. This separation region induced a "jetting effect" on the upstream side of the pipe, which was found to influence the jet exit velocity profile that enters into the cross-flow.

3. The genesis of the counter-rotating vortex pair (CRVP) of the jet was examined and was found to be attributed to two sources. The first was the streamwise vorticity from the jet pipe walls entering into the cross-flow and the second was the streamwise vorticity generated near the lateral edges of the jet exit as the jet shears the cross-flow fluid. In both cases, the streamwise vorticity had the same sense as the CRVP that

was observed in the jet far-field. A pair of small vortices rotating with opposite sense to each other were formed near the lateral sides of the jet exit, and as the vorticity was transported downstream, these vortical structures grow in diameter and become the jet CRVP, which is the dominant feature visible in the time-averaged flow field. Due to mutual vortex induction, the two vortices move closer together laterally until they are in contact, and also move vertically away from the wall (jet lift-off). The present observations agree with similar previous studies, such as Walters and Leylek [25] and Peterson and Plesniak [12].

4. The micro-ramp generated strong counter-rotating vortices of opposite sense to the CRVP. The vorticity in the CRVP was decreased owing to vorticity cancellation from the micro-ramp vortices. The micro-ramp vortices were located close to the flat plate surface where they induce a downwash effect at the midspan. The weakened CRVP of the jet was pushed up above the micro-ramp vortices. Both the micro-ramp and jet vortices are found to persist in the jet far-field.

5. The mean temperature field indicated that the micro-ramp created a larger lateral spread of the coolant. This was caused by the transport of coolant by the counter-rotating vortices from the micro-ramp toward the wall (downwash effect) and laterally along the wall.

6. The micro-ramp produced improvements in both centerline and span-averaged film-cooling effectiveness. In both metrics, it was observed that the curves peak just downstream of the micro-ramp and decay thereafter. This decay was due to the weakening of the micro-ramp vortices, which cannot sustain the transport of coolant to the wall, resulting in decreased cooling effectiveness.

7. Horseshoe vortices were not observed at the jet leading-edge, possibly due to the shallow inclination (20 degrees) of the jet which does not cause roll-up of the cross-flow

boundary layer. No wake vortices were observed since the jet flow remained very close to the wall. The jet separation that was observed at the jet trailing edge suggests the presence of a downstream spiral separation node (DSSN) vortex (as discussed in the previous chapter) but this structure was not resolved in the simulation.

8. Ultimately, more information about the upstream conditions was needed to construct the boundary conditions for the cross-flow and plenum. It is recommended that in future work the experiments and CFD are performed concurrently (if possible) so that they may supplement each other. In addition, it should be noted that there are other (possibly better) ways to generate inflow turbulence than the techniques presented in this work. For example, the cross-flow turbulent boundary-layer could be generated using the recycling method of Lund et al. [117].

Chapter 7

Conclusions and Recommendations

7.1 Conclusions

In the first part of this study, an incompressible Navier-Stokes solver was developed for Graphics Processing Units (GPUs) to explore how GPUs can accelerate performance of existing CFD algorithms. The solver was programmed for the GPU using CUDA, which is a relatively new programming paradigm similar to the C programing language. The GPU-based Navier-Stokes solver was capable of simulating unsteady laminar flows or turbulent flows; for turbulent flows, either a DNS and LES could be performed. The Navier-Stokes equations were solved using the fractional-step method and spatial discretization was performed using the finite volume method on a Cartesian mesh. The pressure-Poisson equation was solved using red-black successive over-relaxation with a geometric multigrid method. The solver algorithm was designed to suit the multithreaded architecture of the GPU, where computational parallelism was required on a per-thread level. The solver performed computations over an order of magnitude faster using a GPU compared with a CPU. In addition, a multi-GPU solver was developed to explore scaling the speed-up even further. This was developed by combining CUDA with POSIX Threads, which is a programming model for parallel computation using multiple/multi-core CPUs in a single machine. The multi-GPU solver delivered sub-linear speed-up scaling relative to the single-GPU solver, which is acceptable given the communication overhead.

An immersed boundary method was developed to handle flow past stationary complex geometries. The boundary of the complex geometry was represented by triangular segments

immersed within the Cartesian mesh. Specifically, the immersed boundary method was a ghost cell method, where velocity and temperature at ghost points within the solid obstacle were set such that desired boundary conditions were satisfied at the immersed boundary. This was achieved by first placing a mirror point in the fluid by mirroring each ghost point along the normal of the closest surface segment. Next, the velocity or temperature values at nearest neighbors to the mirror point were interpolated onto the mirror point. Lastly, ghost point boundary conditions were applied using the mirror point and boundary point. The ghost points for pressure were updated using the pressure-Poission equation, just like the fluid points. As a consequence, local mass conservation was preserved in the ghost cells and there was no mass flux across the immersed boundary.

The GPU-based solver was used to perform Large Eddy Simulations of turbulent flow in film-cooling configurations. This problem is important for modern gas turbine engines, since the film-cooling method is used to protect turbine blades from hot combustion gases. Fundamentally, a film-cooling flow is an inclined jet in a cross-flow, where a jet of coolant flows into a hot cross-flow near a surface to be cooled. The inclination is meant to keep the coolant as close to the surface as possible to maximize cooling. Instead of simulating the film-cooling flow over an actual turbine blade geometry, a simplified geometry was used to extract the fundamental flow physics. This geometry consisted of an inclined jet pipe that intersected a flat plate, where the inclined jet exits at the flat plate surface.

A known problem in the film-cooling method is jet lift-off, were the jet of coolant moves away from the surface due to mutual vortex induction by the counter-rotating vortex pair embedded in the jet. This results in decreased cooling effectiveness at the surface. The primary goal of this study was to examine the effect on the flow field by placing a micro-ramp vortex generator downstream of the jet exit, with particular interest in quantifying changes in cooling performance. The micro-ramp was meant to generate a pair of near-wall counter-rotating vortices of opposite sense to the vortex pair in the jet, resulting in vorticity cancellation and a reduction in jet lift-off from the wall. The present LES study

of film-cooling had two parts and were presented separately in Chapters 5 and 6. In both parts, film-cooling flows with and without a micro-ramp were simulated and compared. The primary difference between the two LES studies is the way the jet exit conditions were implemented. In Chapter 5, the jet exit velocity conditions were generated using a precursor simulation of turbulent pipe flow and the pipe and plenum geometries were not included in the simulation. In Chapter 6, the jet exit velocity conditions were created by including the pipe and plenum in the computational domain, which allowed the jet profile to adjust to the effect of the cross-flow.

In Chapter 5, Large Eddy Simulations of a film-cooling flow were performed with no micro-ramp (baseline case) and with a micro-ramp. These two configurations were simulated using coarse and fine meshes to assess the effect of mesh resolution on the prediction of film-cooling performance. Thus a total of four simulations were performed. The coarse mesh consisted of approximately 8.4 million cells and the fine mesh consisted of approximately 20.5 million cells. The Reynolds number based on the jet diameter and freestream cross-flow velocity was 8000 and the blowing ratio was 1.5. Coolant was injected into the cross-flow at an angle of 35 degrees to the freestream. This problem was designed to be similar to that used by Muldoon and Acharya [19] in order to validate the baseline simulation (no micro-ramp) against their results.

Coherent turbulent structures were identified, including shear-layer vortices on the windward side of the jet, two horseshoe vortices at the jet leading edge (primary vortex driving a weaker secondary vortex), a CRVP in the jet, and DSSN vortices. No wake vortices were found, possibly due to the low blowing ratio. The horseshoe vortex was found to the very beneficial in cooling the wall near the jet leading edge due to entrainment and distribution of jet coolant. For the baseline case, the jet CRVP was the dominant flow structure affecting film-cooling in the jet far-field. Vortex induction by the CRVP caused the jet to lift away from the surface. The vortices were rotating such that an upwash region was generated between the vortices. This upwash caused the coolant to be moved away from the wall;

simultaneously, the upwash caused hot cross-flow fluid to be transported toward the wall. Both of these actions by the upwash contribute to reduced cooling effectiveness.

The micro-ramp generated a pair of counter-rotating vortices of opposite sense to the CRVP in the jet. The micro-ramp vortices helped weaken the jet CRVP thorough vorticity cancellation and created a downwash effect that entrained and transported coolant from the jet toward the wall. The latter effect enhanced film-cooling effectiveness at the wall compared to the baseline case. The vortices generated by the micro-ramp were weak and decayed quickly in the streamwise direction; however, the beneficial downwash effect induced by these vortices persisted downstream. The angle of inclination of the jet was such that the jet flow moved over the micro-ramp, which left only a low-speed flow moving past the micro-ramp in the jet wake. This low-speed flow was responsible for generating the weak vortex pair as it moved past the micro-ramp. Adding a micro-ramp did not appreciably alter the jet trajectory, and jet lift-off was still observed. Despite this, the effect of creating downwash near the wall was sufficient to improve film-cooling effectiveness.

In Chapter 6, Large Eddy Simulations were performed to study the flow in a film-cooling configuration based on wind tunnel experiments conducted at NASA by Zaman et al. [23]. This numerical study has supplemented those experiments by providing information about the film-cooling effectiveness and temperature field, which were not studied in the experiments. The geometry and boundary conditions were prescribed to match the experimental conditions as closely as possible. The jet exit conditions were created by including a plenum chamber and jet pipe upstream of the jet exit, unlike in the previous chapter where a precursor simulation was used for the jet exit. Including the jet pipe in the domain makes the simulation more realistic since this allows the jet exit profile to adjust to the effect of the cross-flow. Two simulations were performed: a baseline case with no micro-ramp and a case with a micro-ramp included one diameter downstream of the jet trailing edge.

For these simulations, the only observable coherent structures were the shear-layer vortices along the windward side of the jet and the CRVP of the jet. Horseshoe vortices were

not observed at the jet leading edge, possibly due to the shallow inclination (20 degrees) of the jet which does not cause roll-up of the cross-flow boundary layer. No wake vortices were observed since the jet flow remained very close to the wall. The flow separation that was observed at the jet trailing edge suggests the presence of a DSSN (downstream spiral separation node) vortex but this structure was not resolved in the simulation.

The genesis of the CRVP of the jet was examined and was found to be attributed to two sources. The first was the streamwise vorticity from the jet pipe walls entering into the cross-flow and the second was the streamwise vorticity generated near the lateral edges of the jet exit as the jet shears the cross-flow fluid. A pair of small vortices rotating with opposite sense to each other were formed near the lateral sides of the jet exit, and as the vorticity was transported downstream these vortical structures grew in diameter and became the jet CRVP, which was the dominant feature visible in the time-averaged flow field. Due to mutual vortex induction, the two vortices moved closer together laterally until they were in contact, and also moved vertically away from the wall (jet lift-off). The present observations agree with similar previous studies, such as Walters and Leylek [25] and Peterson and Plesniak [12].

The flow in the plenum was observed to be almost stagnant, but accelerated as it moved into the jet pipe. A flow separation region developed on the downstream pipe wall owing to the sharp turn at the plenum exit. This separation region induced a "jetting effect" on the upstream side of the pipe, which was a consequence of conservation of mass. The jetting and flow separation were found to influence the jet exit velocity profile, where the largest magnitude in velocity was seen near the upstream edge of the exit hole and the lowest magnitudes were located near the center.

The micro-ramp generated strong counter-rotating vortices of opposite sense to the CRVP. The vorticity in the CRVP was decreased owing to vorticity cancellation from the micro-ramp vortices. The micro-ramp vortices were located close to the flat plate surface where they induced a downwash effect at the midspan. The weakened CRVP of the jet was

pushed up above the micro-ramp vortices. Both the micro-ramp and jet vortices were found to persist in the jet far-field. The mean temperature field indicated that the micro-ramp created a larger lateral spread of the coolant. This was caused by the transport of coolant toward the wall (downwash effect) and then laterally along the wall by the micro-ramp's counter-rotating vortices. The micro-ramp produced improvements in both centerline and span-averaged film-cooling effectiveness. In both metrics, it was observed that the curves peak just downstream of the micro-ramp and decay thereafter. This decay was due to the weakening of the micro-ramp vortices in the streamwise direction.

The LES results were compared to measurements from the experiment of Zaman et al. [23]. The predicted vortex core trajectories in the spanwise direction matched reasonably well versus experiment for the baseline case. The agreement was also good for the micro-ramp case in the near-field, but in the far-field the simulations over-predicted the lateral spreading of the jet. A comparison of the maximum positive mean streamwise vorticity showed reasonable agreement for the baseline case, but the micro-ramp case showed some over-prediction. A comparison of jet penetration for the baseline case showed that the LES predicted an overall shallower slope of the jet than was observed experimentally. For the case with a micro-ramp, the LES predicted more jet lift-off than seen in the experiment.

7.2 Recommendations

For the computational science part of this research, it is recommended that the GPU-based flow solver undergo further optimization. Specifically, the existing data structures for global memory can be arranged such that accesses are coalesced (a concept discussed in Chapter 1), which can lead to decreases in access time. Careful use of shared memory can lead to further reductions in access time. Also, it is recommended that the multi-GPU solver undergo further development, since it never reached the production-ready stage of the single-GPU solver due to time limitations in the research. The development will include extending all

solver features to be multi-GPU compatible and overlapping communication and computation, which can effectively hide the data communication time. In addition, the use of MPI combined with CUDA will be explored, which has the advantage of allowing scaling to larger problem sizes on a cluster. This is an improvement over the present method using POSIX Threads, which is limited to a single workstation.

For the film-cooling simulations presented in Chapter 5, there were limitations to using a precursor simulation. It was found that the mean velocity profile at the jet inflow was somewhat asymmetric, and was possibly due to the jet inflow data being recycled, resulting in a smaller sample size. It is likely that this caused the asymmetry in the jet CRVP that could be observed in the mean flow on streamwise planes. Thus, it is recommended for future studies to either store more precursor data or run the precursor simulation in parallel with the main simulation and communicate the jet inflow data every time-step; either way, the statistics in the jet inflow will improve. Of course, the best way to construct this simulation is to model the plenum and coolant pipe in the computational domain, which corrects the asymmetry problem and allows the jet profile to adjust to the effect of the cross-flow. This approach was used in Chapter 6, and it is recommended to always include the pipe and plenum if the resolution requirements will allow it.

For the film-cooling simulations presented in Chapter 6, more information about the upstream conditions was needed to correctly construct the boundary conditions for the cross-flow and plenum. This became apparent in light of the disagreement between the LES predictions and experimental measurements. It is recommended that in future work the experiments and CFD are performed concurrently (if possible) to ensure that all necessary data for the construction of boundary conditions is available. Lastly, it should be emphasized that only one micro-ramp shape was considered in the present simulations, which was the H0(R0) model used by Zaman et al. [23]. In addition, only one location of the micro-ramp was considered, which was one diameter downstream of the jet. It is recommended that future studies examine alternative shapes and placements of the micro-ramp in an effort to

find a configuration that results in optimal film-cooling effectiveness.

Appendix A

Spatial Discretization of Convection and Diffusion Terms for Momentum Equation

In this Appendix the spatial discretization of the convection and diffusion terms of the u-momentum equation will be shown. The derivation of the convection and diffusion terms of the v- and $w-$ momentum equations are analogous. For the convective term, a flux limiter is incorporated to avoid spurious oscillations from central differencing when the cell Peclet number exceeds the stability limit ($Pe_{cell} < 2$). The flux limiter selected was van Leer's monotonized central (MC) limiter [112, 113]. This limiter works well for a wide range of problems, as noted by LeVeque [118].

We begin with the convection term, derived from Chapter 3,

$$C_u = \int_{\partial\Omega} \rho u(\mathbf{u} \cdot \mathbf{n}) \, dA \tag{A.1}$$

where the boundary $\partial\Omega$ is the union of the six faces of the u-CV (as shown in Figure 3.3 in Chapter 3). Thus this can be written as the sum of intergrals over each face of the u-CV:

$$C_u = \sum_{faces} \int_{A_{face}} \rho u(\mathbf{u} \cdot \mathbf{n}) \, dA = \sum_{faces} F^c_{face} \tag{A.2}$$

where F^c_{face} is the convective flux at a given face. The velocity is taken as a constant along a face, so the convective flux at a face can be written as

$$F^c_{face} = \int_{A_{face}} \rho u(\mathbf{u} \cdot \mathbf{n}) \, dA = u_{face} \int_{A_{face}} \rho(\mathbf{u} \cdot \mathbf{n}) \, dA = u_{face} \dot{m}_{face} \tag{A.3}$$

The convective flux will be numerically approximated using both a low-order numerical flux F^L_{face} and a high-order numerical flux F^H_{face}. The low and high order fluxes are hybridized using a flux limiter function $\phi(r)$ to give the numerical convective flux at a face $\widehat{F}^c_{face}$ [119], written as

$$\widehat{F}^c_{face} = F^L_{face} - \phi(r)[F^L_{face} - F^H_{face}] \tag{A.4}$$

where the flux limiter is the monotonized central limiter

$$\phi(r) = \max[0, \min(2r, 0.5(1+r), 2)] \tag{A.5}$$

and r is the ratio of successive velocity gradients. The high-order flux is used in smooth regions of the solution and is calculated using second-order accurate central differencing. The low-order flux is used in regions of sharp gradients and is calculated using upwinding. In smooth regions the limiter function approaches unity and near sharp gradients the limiter function approaches zero.

Using Equation (A.3) we compute the low-order fluxes for all six faces of the u-CV, where the velocities are obtained via upwinding:

$$F^L_{x+} = \dot{m}_{x+} u_{x+},\ u_{x+} = \begin{cases} u_{i,j,k} & \text{if } u_{i,j,k} > 0 \\ u_{i+1,j,k} & \text{if } u_{i+1,j,k} < 0 \end{cases} \tag{A.6}$$

$$F^L_{x-} = \dot{m}_{x-} u_{x-},\ u_{x-} = \begin{cases} u_{i-1,j,k} & \text{if } u_{i-1,j,k} > 0 \\ u_{i,j,k} & \text{if } u_{i,j,k} < 0 \end{cases} \tag{A.7}$$

$$F^L_{y+} = \dot{m}_{y+} u_{y+},\ u_{y+} = \begin{cases} u_{i,j,k} & \text{if } v_{i,j,k} > 0 \\ u_{i,j+1,k} & \text{if } v_{i,j,k} < 0 \end{cases} \tag{A.8}$$

$$F^L_{y-} = \dot{m}_{y-} u_{y-},\ u_{y-} = \begin{cases} u_{i,j-1,k} & \text{if } v_{i,j-1,k} > 0 \\ u_{i,j,k} & \text{if } v_{i,j-1,k} < 0 \end{cases} \tag{A.9}$$

$$F_{z+}^L = \dot{m}_{z+} u_{z+},\ u_{z+} = \begin{cases} u_{i,j,k+1} & \text{if } (w_{i,j,k} + w_{i+1,j,k})/2 < 0 \\ u_{i,j,k} & \text{if } (w_{i,j,k} + w_{i+1,j,k})/2 > 0 \end{cases} \tag{A.10}$$

$$F_{z-}^L = \dot{m}_{z-} u_{z-},\ u_{z-} = \begin{cases} u_{i,j,k} & \text{if } (w_{i,j,k-1} + w_{i+1,j,k-1})/2 < 0 \\ u_{i,j,k-1} & \text{if } (w_{i,j,k-1} + w_{i+1,j,k-1})/2 > 0 \end{cases} \tag{A.11}$$

Now we can compute the high-order fluxes, where the velocity is obtain via central differencing (or, strictly speaking, linear interpolation):

$$F_{x+}^H = \dot{m}_{x+} u_{x+} = \dot{m}_{x+} \left[\frac{u_{i+1,j,k} + u_{i,j,k}}{2} \right] \tag{A.12}$$

$$F_{x-}^H = \dot{m}_{x-} u_{x-} = \dot{m}_{x-} \left[\frac{u_{i,j,k} + u_{i-1,j,k}}{2} \right] \tag{A.13}$$

$$F_{y+}^H = \dot{m}_{y+} u_{y+} = \dot{m}_{y+} \left[\lambda_{y+}(u_{i,j+1,k}) + (1 - \lambda_{y+}) u_{i,j,k} \right] \tag{A.14}$$

$$F_{y-}^H = \dot{m}_{y-} u_{y-} = \dot{m}_{y-} \left[\lambda_{y-}(u_{i,j,k}) + (1 - \lambda_{y-}) u_{i,j-1,k} \right] \tag{A.15}$$

$$F_{z+}^H = \dot{m}_{z+} u_{z+} = \dot{m}_{z+} \left[\lambda_{z+}(u_{i,j,k+1}) + (1 - \lambda_{z+}) u_{i,j,k} \right] \tag{A.16}$$

$$F_{z-}^H = \dot{m}_{z-} u_{z-} = \dot{m}_{z-} \left[\lambda_{z-}(u_{i,j,k}) + (1 - \lambda_{z-}) u_{i,j,k-1} \right] \tag{A.17}$$

where the linear interpolation factors are

$$\lambda_{y+} = \frac{\Delta y_j}{\Delta y_j + \Delta y_{j+1}}, \quad \lambda_{y-} = \frac{\Delta y_{j-1}}{\Delta y_{j-1} + \Delta y_j}, \quad \lambda_{z+} = \frac{\Delta z_k}{\Delta z_k + \Delta z_{k+1}}, \quad \lambda_{z-} = \frac{\Delta z_{k-1}}{\Delta z_{k-1} + \Delta z_k} \tag{A.18}$$

The low-order and high-order fluxes are now hybridized via Equation (A.4) for all six faces as follows

$$\widehat{F}_{x+}^c = F_{x+}^L - \phi(r_i)[F_{x+}^L - F_{x+}^H] \tag{A.19}$$

$$\widehat{F}_{x-}^c = F_{x-}^L - \phi(r_{i-1})[F_{x-}^L - F_{x-}^H] \tag{A.20}$$

$$\widehat{F}_{y+}^c = F_{y+}^L - \phi(r_j)[F_{y+}^L - F_{y+}^H] \tag{A.21}$$

$$\widehat{F}^c_{y-} = F^L_{y-} - \phi(r_{j-1})[F^L_{y-} - F^H_{y-}] \tag{A.22}$$

$$\widehat{F}^c_{z+} = F^L_{z+} - \phi(r_k)[F^L_{z+} - F^H_{z+}] \tag{A.23}$$

$$\widehat{F}^c_{z-} = F^L_{z-} - \phi(r_{k-1})[F^L_{z-} - F^H_{z-}] \tag{A.24}$$

where the successive velocity gradient ratios are

$$r_i = \frac{u_{i,j,k} - u_{i-1,j,k}}{u_{i+1,j,k} - u_{i,j,k}} \qquad r_j = \frac{u_{i,j,k} - u_{i,j-1,k}}{u_{i,j+1,k} - u_{i,j,k}} \qquad r_k = \frac{u_{i,j,k} - u_{i,j,k-1}}{u_{i,j,k+1} - u_{i,j,k}} \tag{A.25}$$

The numerical convective fluxes from Equations (A.19) to (A.24) are now combined using Equation (A.2) to give the final result

$$\boxed{C_u = \sum_{faces} F^c_{face} \approx \widehat{F}^c_{x+} + \widehat{F}^c_{x-} + \widehat{F}^c_{y+} + \widehat{F}^c_{y-} + \widehat{F}^c_{z+} + \widehat{F}^c_{z-}} \tag{A.26}$$

The diffusion term derived from Chapter 3 is

$$D_u = \int_{\partial\Omega} \mu \nabla u \cdot \mathbf{n} \; dA \tag{A.27}$$

which can be written as the sum of intergrals over each face of the u-CV as

$$D_u = \sum_{faces} \int_{A_{face}} \mu \nabla u \cdot \mathbf{n} \; dA = \sum_{faces} F^d_{face} \tag{A.28}$$

where F^d_{face} is the diffusive flux at a given face. The diffusive flux can be evaluated for a given face as

$$F^d_{face} = \int_{A_{face}} \mu \nabla u \cdot \mathbf{n} \; dA = \mu_{face} (\nabla u \cdot \mathbf{n})|_{face} A_{face} \tag{A.29}$$

Using this equation, the diffusive fluxes can be approximated as follows, yielding the numer-

ical diffusive fluxes

$$F^d_{x+} = \mu_{x+}(\nabla u \cdot \mathbf{i})|_{x+} A_{x+} = \mu_{x+} \left.\frac{\partial u}{\partial x}\right|_{x_+} A_{x+} \approx \mu_{x+} \frac{u_{i+1,j,k} - u_{i,j,k}}{\Delta x_{i+1}} A_{x+} \equiv \widehat{F}^d_{x+} \tag{A.30}$$

$$F^d_{x-} = \mu_{x-}(\nabla u \cdot -\mathbf{i})|_{x-} A_{x-} = -\mu_{x-} \left.\frac{\partial u}{\partial x}\right|_{x_-} A_{x-} \approx -\mu_{x-} \frac{u_{i,j,k} - u_{i-1,j,k}}{\Delta x_i} A_{x-} \equiv \widehat{F}^d_{x-} \tag{A.31}$$

$$F^d_{y+} = \mu_{y+}(\nabla u \cdot \mathbf{j})|_{y+} A_{y+} = \mu_{y+} \left.\frac{\partial u}{\partial y}\right|_{y_+} A_{y+} \approx \mu_{y+} \frac{u_{i,j+1,k} - u_{i,j,k}}{(\Delta y_{j+1} + \Delta y_j)/2} A_{y+} \equiv \widehat{F}^d_{y+} \tag{A.32}$$

$$F^d_{y-} = \mu_{y-}(\nabla u \cdot -\mathbf{j})|_{y-} A_{y-} = -\mu_{y-} \left.\frac{\partial u}{\partial y}\right|_{y_-} A_{y-} \approx -\mu_{y-} \frac{u_{i,j,k} - u_{i,j-1,k}}{(\Delta y_j + \Delta y_{j-1})/2} A_{y-} \equiv \widehat{F}^d_{y-} \tag{A.33}$$

$$F^d_{z+} = \mu_{z+}(\nabla u \cdot \mathbf{k})|_{z+} A_{z+} = \mu_{z+} \left.\frac{\partial u}{\partial z}\right|_{z_+} A_{z+} \approx \mu_{z+} \frac{u_{i,j,k+1} - u_{i,j,k}}{(\Delta z_{k+1} + \Delta z_k)/2} A_{z+} \equiv \widehat{F}^d_{z+} \tag{A.34}$$

$$F^d_{z-} = \mu_{z-}(\nabla u \cdot -\mathbf{k})|_{z-} A_{z-} = -\mu_{z-} \left.\frac{\partial u}{\partial z}\right|_{z_-} A_{z-} \approx -\mu_{z-} \frac{u_{i,j,k} - u_{i,j,k-1}}{(\Delta z_k + \Delta z_{k-1})/2} A_{z-} \equiv \widehat{F}^d_{z-} \tag{A.35}$$

The numerical diffusive fluxes from Equations (A.30) to (A.35) are now combined using Equation (A.28) to give the final result

$$\boxed{D_u = \sum_{faces} F^d_{face} \approx \widehat{F}^d_{x+} + \widehat{F}^d_{x-} + \widehat{F}^d_{y+} + \widehat{F}^d_{y-} + \widehat{F}^d_{z+} + \widehat{F}^d_{z-}} \tag{A.36}$$

Appendix B

Equivalence of time-steps

The following calculation shows the physical time-step for the film-cooling simulations presented in Chapter 5 is the same as the physical time-step for the precursor LES of turbulent pipe flow. Let the subscript p denote the precursor simulation and subscript f denote the film-cooling simulation. The time-steps for the simulations were

$$\Delta t_p = 0.0001(d/u_\tau) \tag{B.1}$$

$$\Delta t_f = 0.001(d/u_\infty) \tag{B.2}$$

and the Reynolds numbers were

$$Re_\tau = u_\tau d/\nu = 764 \tag{B.3}$$

$$Re_\infty = u_\infty d/\nu = 8000 \tag{B.4}$$

Dividing Equation (B.3) by Equation (B.4) yields

$$u_\tau/u_\infty = 764/8000 \tag{B.5}$$

or

$$u_\tau = (764/8000)u_\infty \tag{B.6}$$

Substituting Equation (B.6) into Equation (B.1) yields

$$\Delta t_p = 0.0001(d/u_\tau) = 0.0001\frac{d}{(764/8000)u_\infty} \tag{B.7}$$

or

$$\Delta t_p \approx 0.001(d/u_\infty) \tag{B.8}$$

and thus

$$\Delta t_p \approx \Delta t_f \tag{B.9}$$

References

[1] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. The MIT Press, Cambridge, MA, 1996.

[2] D. R. Butenhof. *Programming with POSIX threads*. Addison-Wesley, Boston, MA, 1997.

[3] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. The MIT Press, Cambridge, MA, 2008.

[4] NVIDIA. NVIDIA CUDA C Programming Guide. Version 3.2, 2010.

[5] S. B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[6] D. B. Kirk and W. W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers, Burlington, MA, 2010.

[7] NVIDIA. CUDA C Best Practices Guide. Version 3.2, 2010.

[8] R. S. Bunker. A review of shaped hole turbine film-cooling technology. *Journal of Heat Transfer*, 127:441–453, 2005.

[9] D. G. Hyams and J. H. Leylek. A detailed analysis of film cooling physics: part III–streamwise injection with shaped holes. *Journal of Turbomachinery*, 122:122–132, 2000.

[10] J. Andreopoulos and W. Rodi. Experimental investigation of jets in a crossflow. *J. Fluid Mech.*, 138:93–127, 1984.

[11] T. F. Fric and A. Roshko. Vortical structure in the wake of a transverse jet. *J. Fluid Mech.*, 279:1–47, 1994.

[12] S. D. Peterson and M. W. Plesniak. Evolution of jets emanating from short holes into crossflow. *J. Fluid Mech.*, 503:57–91, 2004.

[13] L. K. Su and M. G. Mungal. Simultaneous measurements of scalar and velocity field evolution in turbulent crossflowing jets. *J. Fluid Mech.*, 513:1–45, 2004.

[14] K. B. M. Q. Zaman and J. K. Foss. The effect of vortex generators on a jet in a cross-flow. *Phys. Fluids*, 9(1), 1997.

[15] S. Muppidi and K. Mahesh. Study of trajectories of jets in crossflow using direct numerical simulations. *J. Fluid Mech.*, 530:81–100, 2005.

[16] S. Muppidi and K. Mahesh. Direct numerical simulation of round turbulent jets in crossflow. *J. Fluid Mech.*, 574:59–84, 2007.

[17] L. L. Yuan, R. L. Street, and J. H. Ferziger. Large-eddy simulations of a round jet in crossflow. *J. Fluid Mech.*, 379:71–104, 1999.

[18] J. D. Heidmann and S. Ekkad. A novel antivortex turbine film-cooling hole concept. *Journal of Turbomachinery*, 130, 2008.

[19] F. Muldoon and S. Acharya. DNS study of pulsed film cooling for enhanced cooling effectiveness. *International Journal of Heat and Mass Transfer*, 52:3118–3127, 2009.

[20] S. M. Coulthard, R. J. Volino, and K. A. Flack. Effect of jet pulsing on film cooling—part I: effectiveness and flow-field temperature results. *Journal of Turbomachinery*, 129(2):232–246, 2007.

[21] S. M. Coulthard, R. J. Volino, and K. A. Flack. Effect of jet pulsing on film cooling—part II: heat transfer results. *Journal of Turbomachinery*, 129(2):247–257, 2007.

[22] D. L. Rigby and J. D. Heidmann. Improved film cooling effectiveness by placing a vortex generator downstream of each hole. In *Proceedings of ASME Turbo Expo 2008*, 2008.

[23] K. B. M. Q. Zaman, D. L. Rigby, and J. D. Heidmann. Experimental study of an inclined jet-in-cross-flow interacting with a vortex generator. In *48th AIAA Aerospace Sciences Meeting*, 2010.

[24] S. Dhungel, A. Phillips, S. V. Ekkad, and J. D. Heidmann. Experimental investigation of a novel anti-vortex film cooling hole design. *Proceedings of ASME Turbo Expo 2007*, 2007.

[25] D. K. Walters and J. H. Leylek. A detailed analysis of film-cooling physics: part I—streamwise injection with cylindrical holes. *Journal of Turbomachinery*, 122:102–112, 2000.

[26] A. Kohli and D. G. Bogard. Improving film cooling performance using airfoil contouring. *Journal of Turbomachinery*, 130, 2008.

[27] A. Kohli and K. A. Thole. A CFD investigation on the effects of entrance crossflow directions to film-cooling holes. In *National Heat Transfer Conference*, 1997.

[28] S. Acharya, M. Tyagi, and A. Hoda. Flow and heat transfer predictions for film cooling. In *Heat transfer in gas turbine systems, Ann. N.Y. Acad. Sci.*, volume 934, pages 110–125, 2001.

[29] O. Kartuzova, D. Danila, M. B. Ibrahim, and R. J. Volino. Computational simulation of cylindrical film hole with jet pulsation on flat plates. *Journal of Propulsion and Power*, 25(6):1249–1258, 2009.

[30] M. Tyagi and S. Acharya. Large eddy simulation of film cooling flow from an inclined cylindrical jet. *Journal of Turbomachinery*, 125:734–742, 2003.

[31] P. Renze, M. Meinke, and W. Schröder. LES of turbulent mixing in film cooling flows. In *Conference on Turbulence and Interactions*, 2006.

[32] Y. V. Peet and S. K. Lele. Near field of film cooling jet issued into a flat plate boundary layer: LES study. In *Proceedings of ASME Turbo Expo 2008*, 2008.

[33] F. Muldoon and S. Acharya. Analysis of $k - \epsilon$ budgets for film cooling using direct numerical simulation. *AIAA Journal*, 44(12):3010–3021, 2006.

[34] C. E. Scheidegger, J. L. D. Comba, and R. D. da Cunha. Practical CFD simulations on programmable graphics hardware using SMAC. *Computer Graphics forum*, 24:715–728, 2005.

[35] E. Elsen, P. LeGresley, and E. Darve. Large calculation of the flow over a hypersonic vehicle using a GPU. *Journal of Computational Physics*, 227(24):10148–10161, 2008.

[36] T. Brandvik and G. Pullan. Acceleration of a 3D Euler solver using commodity graphics hardware. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA 2008-607, January 7-10 2008, Reno, Nevada.

[37] J. M. Cohen and M. J. Molemaker. A fast double precision CFD code using CUDA. In *21st International Conference on Parallel Computational Fluid Dynamics*, 2009.

[38] A. F. Shinn and S. P. Vanka. Implementation of a semi-implicit pressure-based multigrid fluid flow algorithm on a graphics processing unit. In *Proceedings of the ASME 2009 IMECE*, 2009.

[39] A. F. Shinn, S. P. Vanka, and W. W. Hwu. Direct numerical simulation of turbulent flow in a square duct using a graphics processing unit (GPU). In *40th AIAA Fluid Dynamics Conference*, 2010.

[40] R. Chaudhary, S. P. Vanka, and B. G. Thomas. Direct numerical simulations of magnetic field effects on turbulent flow in a square duct. *Phys. Fluids*, 22(7), 2010.

[41] J. Thibault and I. Senocak. CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows. In *47th AIAA Aerospace Sciences Meeting*, AIAA 2009-758, January 5-8 2009.

[42] M. Griebel and P. Zaspel. A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations. *Comput. Sci. Res. Dev.*, 25:65–73, 2010.

[43] W. Li, Z. Fan, X. Wei, and A. Kaufman. GPU-based flow simulation with complex boundaries. In *GPU Gems II*, chapter 47. Addison Wesley, 2005.

[44] J. Tölke. Implementation of a Lattice Boltzmann kernel using the compute unified device architecture developed by NVIDIA. *Computing and Visualization in Science*, 2008.

[45] L. Peng, K. Nomura, T. Oyakawa, R. Kalia, A. Nakano, and P. Vashishta. Parallel Lattice Boltzmann flow simulation on emerging multi-core platforms. In *Euro-Par 2008 - Parallel Processing*, volume 5168 of *Lecture Notes in Computer Science*, pages 763–777. Springer Berlin / Heidelberg, 2008.

[46] D. Marsh. Molecular dynamics-Lattice Boltzmann hybrid method on graphics processors. Master's thesis, University of Illinois at Urbana-Champaign, 2010.

[47] S. P. Vanka, A. F. Shinn, and K. C. Sahu. Computational fluid dynamics using graphics processing units: Challenges and opportunities. In *Proceedings of the ASME 2011 IMECE*, 2011.

[48] R. J. Margason. Fifty years of jet in crossflow research. In *AGARD Symp. on a Jet in CrossFlow*, number AGARD CP-534, Winchester, UK, 1993.

[49] D. G. Bogard and K. A. Thole. Gas turbine film cooling. *Journal of Propulsion and Power*, 22(2), 2006.

[50] R. J. Goldstein. Film cooling. In *Advances in Heat Transfer*, volume 7, pages 321–379, San Diego, 1971. Academic Press.

[51] D. M. Kercher. A film-cooling CFD bibliography: 1971-1996. *International Journal of Rotating Machinery*, 4(1):61–72, 1998.

[52] S. A. Sherif and R. H. Pletcher. Measurements of the flow and turbulence characteristics of round jets in crossflow. *J. Fluids Engineering*, 111:165–171, 1989.

[53] A. K. Sinha, D. G. Bogard, and M. E. Crawford. Film-cooling effectiveness downstream of a single row of holes with variable density ratio. *Journal of Turbomachinery*, 113:442–449, 1991.

[54] K. T. McGovern and J. H. Leylek. A detailed analysis of film cooling physics: part II–Compound-angle injection with cylindrical holes. *Journal of Turbomachinery*, 122:113–121, 2000.

[55] R. A. Brittingham and J. H. Leylek. A detailed analysis of film cooling physics: part IV–Compound-angle injection with shaped holes. *Journal of Turbomachinery*, 122:133–145, 2000.

[56] S. Na and T. Shih. Increasing adiabatic film-cooling effectiveness by using an upstream ramp. *Journal of Heat Transfer*, 129:464–471, 2007.

[57] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.

[58] G. Iaccarino and R. Verzicco. Immersed boundary technique for turbulent flow simulations. *Appl. Mech. Rev.*, 56:331–347, 2003.

[59] C. S. Peskin. *Flow Patterns Around Heart Valves: A Digital Computer Method for Solving the Equations of Motion.* PhD thesis, Albert Einstein College of Medicine, 1972.

[60] C. S. Peskin. Flow patterns around heart valves: a numerical method. *J. Comput. Phys.*, 10:220–252, 1972.

[61] C. S. Peskin. Numerical analysis of blood flow in the heart. *J. Comput. Phys.*, 25:220–252, 1977.

[62] D. M. McQueen and C. S. Peskin. A three-dimensional computational method for blood flow in the heart. II. Contractile fibers. *J. Comput. Phys.*, 82, 1989.

[63] D. M. McQueen and C. S. Peskin. Shared-memory parallel vector implementation of the immersed boundary method for the computation of blood flow in the beating mammalian heart. *J. Supercomp.*, 1997.

[64] D. Goldstein, R. Handler, and L. Sirovich. Modeling a no-slip flow boundary with an external force field. *J. Comput. Phys.*, 105:354–366, 1993.

[65] J. Mohd-Yusof. Combined immersed boundary/b-spline methods for simulations of flows in complex geometries. In *Annual Research Briefs*, pages 317–327. NASA Ames Research Center/Stanford University Center for Turbulence Research, 1997.

[66] E. A. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof. Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *J. Comput. Phys.*, 161:35–60, 2000.

[67] J. Choi, R. C. Oberoi, J. R. Edwards, and J. A. Rosati. An immersed boundary method for complex incompressible flows. *J. Comput. Phys.*, 224:757–784, 2007.

[68] S. Ghosh, J. Choi, and J. R. Edwards. RANS and hybrid LES/RANS simulations of the effects of micro vortex generators using immersed boundary methods. In *38th AIAA Fluid Dynamics Conference*, 2008.

[69] T. Gao, Y. Tseng, and X. Lu. An improved hybrid cartesian/immersed boundary method for fluid–solid flows. *Int. J. Numer. Meth. Fluids*, 55:1189–1211, 2007.

[70] J. Kim, D. Kim, and H. Choi. An immersed-boundary finite-volume method for simulations of flow in complex geometries. *J. Comput. Phys.*, 171:132–150, 2001.

[71] D. Kim and H. Choi. Immersed boundary method for flow around an arbitrarily moving body. *J. Comput. Phys.*, 212:662–680, 2006.

[72] M. Tyagi and S. Acharya. Large eddy simulation of turbulent flows in complex and moving rigid geometries using the immersed boundary method. *Int. J. Numer. Meth. Fluids*, 48:691–722, 2005.

[73] Y. Tseng and J. Ferziger. A ghost-cell immersed boundary method for flow in complex geometry. *J. Comput. Phys.*, 192(2):593–623, 2003.

[74] A. Mark and B.G.M. van Wachem. Derivation and validation of a novel implicit second-order accurate immersed boundary method. *J. Comput. Phys.*, 227:6660–6680, 2008.

[75] R. Mittal, H. Dong, M. Bozkurttas, F. M. Najjar, A. Vargas, and A. von Loebbecke. A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *J. Comput. Phys.*, 227:4825–4852, 2008.

[76] A. F. Shinn, M. A. Goodwin, and S. P. Vanka. Immersed boundary computations of shear- and buoyancy-driven flows in complex enclosures. *International Journal of Heat and Mass Transfer*, 52:4082–4089, 2009.

[77] R. P. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys.*, 152:457–492, 1999.

[78] R. P. Fedkiw, T. Aslam, and S. Xu. The ghost fluid method for deflagration and detonation discontinuities. *J. Comput. Phys.*, 154:393–427, 1999.

[79] R. J. LeVeque and Z. Li. The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *SIAM J. Numer. Anal.*, 31:1019–1044, 1994.

[80] R. J. LeVeque and Z. Li. Immersed interface methods for Stokes flow with elastic boundaries or surface tension. *SIAM J. Sci. Comput.*, 18:709–735, 1997.

[81] Z. Li and M. Lai. The immersed interface method for the navier-stokes equations with singular forces. *J. Comput. Phys.*, 171:822–842, 2001.

[82] L. Lee and R. LeVeque. An immersed interface method for incompressible navier-stokes equations. *SIAM J. Sci. Comput.*, 25(3):832–856, 2003.

[83] S. Xu and Z. J. Wang. Systematic derivation of jump conditions for the immersed interface method in three-dimensional flow simulation. *SIAM J. Sci. Comput.*, 27(6):1948–1980, 2006.

[84] S. Xu and Z. J. Wang. A 3d immersed interface method for fluid–solid interaction. *Comput. Methods Appl. Mech. Engrg.*, 197:2068–2086, 2008.

[85] K. Karagiozis, R. Kamakoti, and C. Pantano. A low numerical dissipation immersed interface method for the compressible Navier-Stokes equations. *J. Comput. Phys.*, 229:701–727, 2010.

[86] D. Clarke, M. Salas, and H. Hassan. Euler calculations for multi-element airfoils using cartesian grids. *AIAA Journal*, 24:1128–1135, 1986.

[87] T. Ye, R. Mittal, H. S. Udaykumar, and W. Shyy. An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries. *J. Comput. Phys.*, 156:209–240, 1999.

[88] M. P. Kirkpatrick, S. W. Armfield, and J. H. Kent. A representation of curved boundaries for the solution of the Navier-Stokes equations on a staggered three-dimensional cartesian grid. *J. Comput. Phys.*, 184:1–36, 2003.

[89] D. Hartmann, M. Meinke, and W. Schröder. A strictly conservative cartesian cut-cell method for compressible viscous flows on adaptive grids. *Computer Methods in Applied Mechanics and Engineering*, 200(9-12):1038 – 1052, 2011.

[90] R. Mittal, Y. Utturkar, and H. S. Udaykumar. Computational modeling and analysis of biomimetic flight mechanisms. In *40th AIAA Aerospace Sciences Meeting*, 2002.

[91] Y. Utturkar, R. Mittal, P. Rampunggoon, and L. Cattafesta. Sensitivity of synthetic jets to the design of the jet cavity. In *40th AIAA Aerospace Sciences Meeting*, 2002.

[92] Y. V. Peet. *Film Cooling from Inclined Cylindrical Holes using Large Eddy Simulations*. PhD thesis, Stanford University, 2006.

[93] S. Peng and L. Davidson. On a subgrid-scale heat flux model for large eddy simulation of turbulent thermal flow. *International Journal of Heat and Mass Transfer*, 45(7):1393–1405, 2002.

[94] J. Blazek. *Computational Fluid Dynamics: Principles and Applications*, volume 1. Elsevier, 2nd edition, 2005.

[95] J. Smagorinsky. General circulation experiments with the primitive equations I. The basic experiment. *Monthly Weather Review*, 91(3):99–164, 1963.

[96] F. Nicoud and F. Ducros. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. *Flow, Turbulence and Combustion*, 62:183–200, 1999.

[97] F.H. Harlow and J.E. Welch. Numerical calculation of time dependent viscous incompressible flow with free surface. *Phys. Fluids*, 8:2182–2189, 1965.

[98] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22, 1968.

[99] R. Témam. Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires (i). *Archive for Rational Mechanics and Analysis*, 32(2):135–153, 1969.

[100] J. H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics*. Springer, Berlin, Germany, 3rd edition, 2002.

[101] B. Barney. POSIX Threads Programming. Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/pthreads/, 2010.

[102] A. Gilmanov, F. Sotiropoulos, and E. Balaras. A general reconstruction algorithm for simulating flows with complex 3D immersed boundaries on Cartesian grids. *J. Comp. Phys.*, 191:660–669, 2003.

[103] H. Ku, R. Hirsh, and T. Taylor. A pseudospectral method for solution of the three-dimensional incompressible Navier-Stokes equations. *J. Comput. Phys.*, 70:439–462, 1987.

[104] R.K. Madabhushi and S.P. Vanka. Large eddy simulation of turbulence-driven secondary flow in a square duct. *Phys. Fluids*, 3(11):2734–2745, 1991.

[105] A. Huser and S. Biringen. Direct numerical simulation of turbulent flow in a square duct. *J. Fluid Mech.*, 257:65–95, 1993.

[106] S. Gavrilakis. Numerical simulation of low-reynolds-number turbulent flow through a straight square duct. *J. Fluid Mech.*, 244:101–129, 1992.

[107] J. G. M. Eggels, F. Unger, M. H. Weiss, J. Westerweel, R. J. Adrian, R. Friedrich, and F. T. M. Nieuwstadt. Fully developed turbulent pipe flow: A comparison between direct numerical simulation and experiment. *J. Fluid Mech.*, 268:175–209, 1994.

[108] A. Sohankar, C. Norberg, and L. Davidson. Low-reynolds-number flow around a square cylinder at incidence: study of blockage, onset of vortex shedding and outlet boundary condition. *International Journal for Numerical Methods in Fluids*, 26:39–56, 1998.

[109] T. Yoshida, T. Watanabe, and I. Nakamura. Numerical analysis of open boundary conditions for incompressible viscous flow past a square cylinder. *Trans. JSME*, 59:2799–2806, 1993.

[110] J. M. J. den Toonder and F. T. M. Nieuwstadt. Reynolds number effects in a turbulent pipe flow for low to moderate Re. *Phys. Fluids*, 9(11):3398–3409, 1997.

[111] Tecplot, Inc. *Extracting vortex cores, Tecplot 360 2009 User's Manual.*

[112] B. van Leer. Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme. *J. Comput. Phys.*, 14:361–370, 1974.

[113] B. van Leer. Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection. *J. Comput. Phys.*, 23:276–299, 1977.

[114] H. Werner and H. Wengle. Large-eddy simulation of turbulent flow over and around a cube in a plate channel. In *8th Symposium on Turbulent Shear Flows*, pages 155–168, 1991.

[115] F. M. White. *Viscous Fluid Flow.* McGraw Hill, 3rd edition, 2006.

[116] L. Howes and D. Thomas. Chapter 37. Efficient Random Number Generation and Application Using CUDA. In H. Nguyen, editor, *GPU Gems 3*. Pearson Education, Inc., 2008.

[117] T. S. Lund, X. Wu, and K. D. Squires. Generation of turbulent inflow data for spatially-developing boundary layer simulations. *J. Comput. Phys.*, 140:233–258, 1998.

[118] R. J. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge University Press, 2002.

[119] R. J. LeVeque. *Numerical methods for conservation laws*. Birkhäuser, Basel, Switzerland, 1992.

Vita

Aaron F. Shinn was born on July 5th, 1982 in Charleston, West Virginia. He graduated from Buffalo High School in Buffalo, West Virginia in 2000. He then attended West Virginia University in Morgantown, West Virginia and graduated Summa Cum Laude with a B.S. in Aerospace Engineering and a B.S. in Mechanical Engineering in Spring 2005. In Fall 2005, he began pursuing a M.S. in Aerospace Engineering at UIUC and was advised by Prof. S. P. Vanka. After graduating with a M.S. in Spring 2007, he stayed at UIUC to pursue a Ph.D. in Mechanical Engineering and continued being advised by Prof. S. P. Vanka. His Ph.D. work was primarily supported by a three-year NASA GSRP fellowship sponsored by NASA Glenn Research Center. Partial support was provided by a CSE Fellowship (jointly sponsored by the CSE Program at UIUC and by NCSA) and endowed fellowships from the MechSE department.